FUJITSU

AN706-00043-1v0-E

32-BIT MICROCONTROLLER
MB9B100A/MB9B300A/MB9B400A/MB9B500A Series

FM3 family inverter reference Firmware

USER MANUAL

ARM™ ARM and Cortex-M3 are the trademarks of ARM Limited in the EU and other countries.

All Rights Reserved.

The contents of this document are subject to change without notice. Customers are advised to consult with FUJITSU sales representatives before ordering.

The information, such as descriptions of function and application circuit examples, in this document are presented solely for the purpose of reference to show examples of operations and uses of Fujitsu semiconductor device; Fujitsu does not warrant proper operation of the device with respect to use based on such information. When you develop equipment incorporating the device based on such information, you must assume any responsibility arising out of such use of the information. Fujitsu assumes no liability for any damages whatsoever arising out of the use of the information.

Any information in this document, including descriptions of function and schematic diagrams, shall not be construed as license of the use or exercise of any intellectual property right, such as patent right or copyright, or any other right of Fujitsu or any third party or does Fujitsu warrant non-infringement of any third-party's intellectual property right or other right by using such information. Fujitsu assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein.

The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).

Please note that Fujitsu will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.

Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.

If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the prior authorization by Japanese government will be required for export of those products from Japan.

The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

Copyright© 2011 FUJITSU SEMICONDUCTOR LIMITED all rights reserved

Revision History

Rev	Date	Remark
1.0	Jul. 13, 2011	First Edition

Table of Contents

Revision History 2
Table of Contents 3
1 Introduction 5
2 MCL Specification 6
2.1 Overview 6
2.2 Features 7
2.3 Peripheral Occupation 7
3 MCL Architecture Introduction 8
4 Data Structure Introduction 9
5 MCL Functions 10
5.1 _Global folder 10
5.2 Driver folder 11
5.2.1 Functions in CoreCtrl folder 11
5.2.2 Functions in PeripheralCtrl folder 15
5.3 Algorithm folder 49
5.3.1 Functions in Control Folder 49
5.3.2 Functions in Math Folder 50
5.3.3 Functions in Transfer Folder 51
5.3.4 Functions in Transformation Folder 53
5.4 Application folder 55
5.4.1 Functions in Init Folder 55
5.4.2 Functions in MotorCtrl Folder 58
5.4.3 Functions in Others Folder 63
6 Global Variables 64
7 User Interface I 68
7.1 Macros on-Chip Drive Interface 68
7.1.1 Interface of Definition Type 68
7.1.2 Interface of Function Type 69
7.2 User Parameters configuration Interface 70
7.2.1 Introduction 70
7.2.2 How to Use 72
7.3 Motor Driving interface 74
8 User Interface II 75
8.1 Macros on-Chip Drive Interface 75

8.2 User Parameters configuration Interface 75
8.3 Macros on-chip Startup Interface 75
8.4 Macros In-chip Stop Interface 75
8.5 Log Warning Interface 76
8.6 Math interface 76
8.7 Transfer function interface 76
8.8 Transformation function interface 77
8.9 Motor Control Algorithm Interface 77
8.10 Others 77
8.11 Definition Interface 78
9 Installation and Usage for MCL 79
9.1 Installation 79
9.2 Usage 86
9.2.1 main() 86
9.2.2 Configuring Peripherals 87
9.2.3 ISR Interface in MCL 87
9.2.4 Debug with EWARM 90
9.2.5 Debug with GUI 93

1 Introduction

This user manual describes how to use the Motor Control Library to develop your firmware in MB9B500 Series.

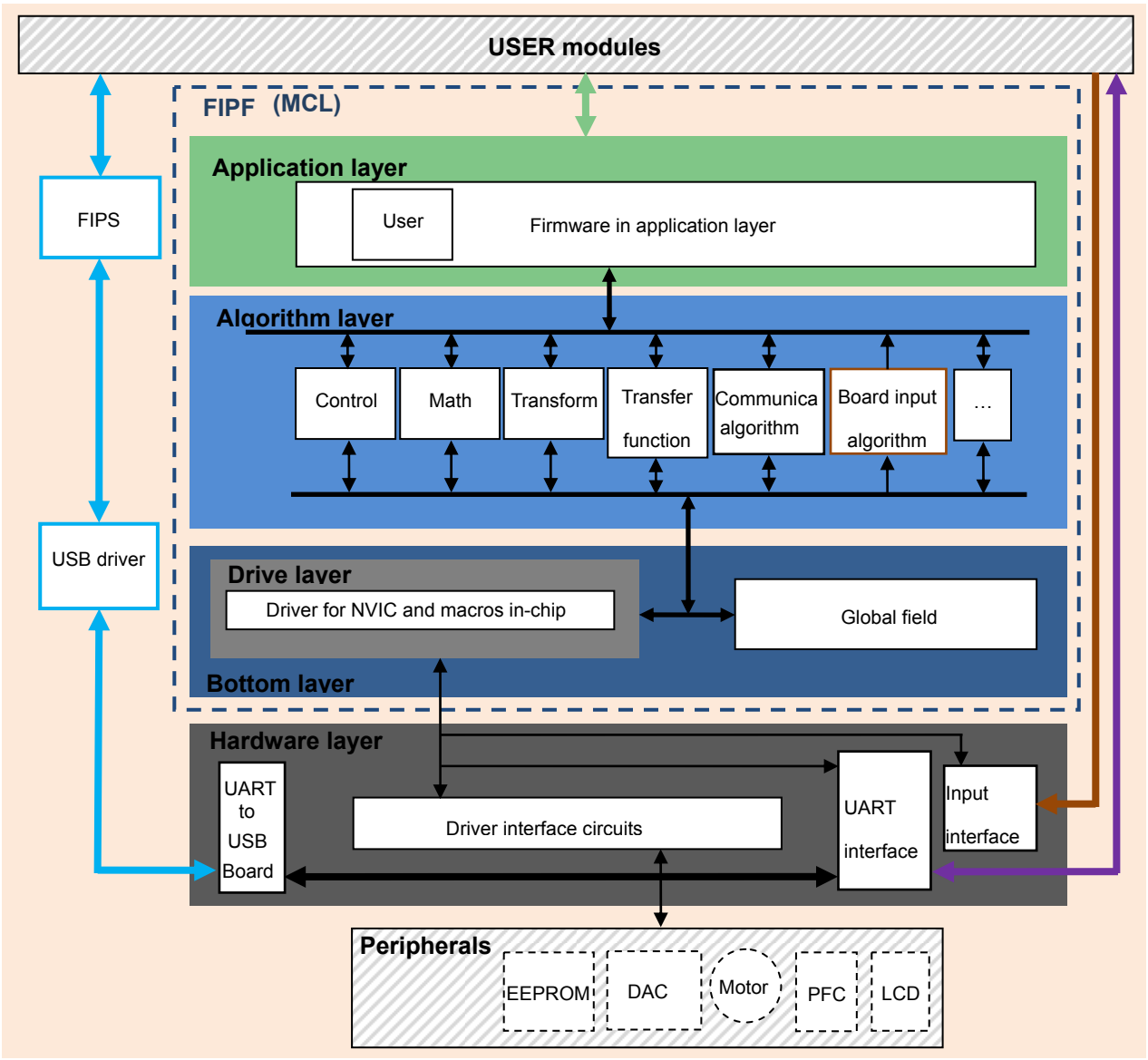
In this document, Motor Control Library is abbreviated to “MCL”.

2 MCL Specification

Introduction of MCL structure

2.1 Overview

MCL is a motor control library which is used to help users developing firmware for motor control. By calling functions in MCL, user can develop motor control firmware easily.



2.2 Features

- Only need to include one header file to access all MCL functions

  MCL consists of two root folders -- **head** and **source**. All the *.h files is placed in the head folder and all the *.c files is placed in the source folder. In the head folder, there is a root header file named *HeadInOne_MCL.h*. By including this file in source file, the source file is able to access any function in MCL.
- MCL functions are divided into 3 layers

  Driver layer – the place to hold functions which directly operate hardware registers.
  Algorithm layer – the place to hold basic inverter control functions.
  Application layer – the place to hold application functions.
- MCL offers 2 ways to access on-chip registers. One is macro definitions which can access registers in a high speed. The other is function which can make the coding easy to maintain. The user can choose one of them according to the requirement. In addition, you also can use these two ways at the same time to access registers.
- MCL includes a user interface to facilitate user's initialization to on-chip macros and configuration of system parameters.
- MCL includes exception checking function to judge automatically whether the user parameters and function parameters are suitable or not.
- Other editor supporting.

  For example, Eclipse /Netbeans /Codeblocks /etc., MCL can eliminate these editor errors and warning prompting.

2.3 Peripheral Occupation

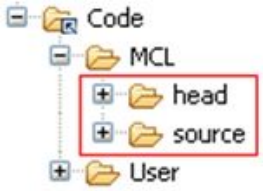
MCL occupies some peripherals. Please refer Table 1 as follows.

Table 1 MCL peripherals occupation

Peripheral	Amount	Details
Dual Timer	1	MCL always uses Dual Timer-ch2.
MFT	1	User can select and configure among FRTs on the chip.
ADC	4 channels	User can select and configure among ADC channels on the chip. (This is In the case of without PFC function.)
	8 channels	User can select and configure among ADC channels on the chip. (This is In the case of with PFC function.)
UART	1	MCL always uses UART-ch6

3 MCL Architecture Introduction

MCL includes 2 root folders – MCL/head and MCL/source. Each of the root folders is divided into 4 parts -- _Global, Driver, Algorithm and Application, as shown in Figure 1. In head folder, there is a header file named HeadInOne_MCL.h. It includes all MCL header files. If users want to call functions in MCL, this file should be included in main.c.



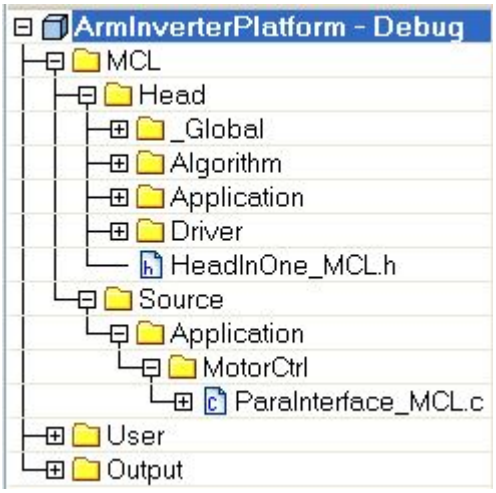


Figure 1 file structure in head and source folder

Source folder is used to hold all source files (*.c files) and head folder is used to hold all header files (*.h files). Source files are a place to hold functions, and header files are a place to do the following jobs.

- macros definition
- data types definition
- global variables definition
- function declaration

The Qn (n = 15) format is wildly used In MCL. Many data are shown as Q15 format in the debug window. The relationship between Q15 value and real value is as follows:

Q15 value = real value × 32768

4 Data Structure Introduction

All data structure in MCL is not included here, only some important ones are listed.

Name	Description
_stWarningCode	Save warning message
_stErrInfo	Save error message
_stSapmle	Relative properties to ADC sampling is stored here
_stDataInThreeAxis	Save data in symmetrical 3-phase coordinate
_stDataInFixAxis	Save data in alpha-beta coordinate
_stDataInRotAxis	Save data in d-q coordinate
_stParaForMcu	MCU parameters given by user is placed at here
_stParaForClock	Clock parameters given by user is placed at here
_stParaForMotor	Motor parameters given by user is placed at here
_stParaForDualTimer2	Dual timer-ch2 parameters given by user is placed at here
_stParaForLPF	LPF parameters given by user is placed at here
_stParaForDCBus	DC bus parameters given by user is placed at here
_stParaForPID	PID parameters given by user is placed at here
_stParaForMotorRunCtrl	Motor control parameters given by user is placed at here
_stInitFRT	FRT parameters given by user is placed at here
_stInitOCCP	OCCP parameters given by user is placed at here
_stInitWFG	WFG parameters given by user is placed at here
_stInitADC	ADC parameters given by user is placed at here
_stInitDTTI	DTTI parameters given by user is placed at here
_stMcuPara	Save MCU parameters given by user
_stDCBusPara	Save DC bus parameters given by user
_stMotorPara	Save motor parameters given by user
_stDualTimer2	Save dual timer-ch2 parameters given by user
_stLPFPara	Save LPF parameters given by user
_stPIDPara	Save PID parameters given by user
_stMotorRunCtrl	Save motor control parameters given by user and save motor running status given by user

5 MCL Functions

All data structure in MCL is not included here, only some important ones are listed.

5.1 _Global folder

These functions are in the folder of MCL/source/_Global/Debug

```
//
// FUNCTION:
// To log the warning code when detect warning. This function will record
// WarningCode into the member of a global structure named gstWarningCode.
// The maximum warning information amount of this function can record is
// MAX_WARNING_CODE_CNT.
//
// PARAMETERS:
// WarningCode - warning code
//
// warning code
// 0x01xx - warnings relating to calculating
// 0x02xx - Warnings relating to MFT
//
// RETURN:
// none
//
void LogWarningCode(const INT16U WarningCode);
```

5.2 Driver folder

5.2.1 Functions in CoreCtrl folder

These functions are in the folder of MCL/source/driver/CoreCtrl

```
//
// FUNCTION:
// Enable IRQ
//
// PARAMETERS:
// NVIC_No - NVIC serial No. corresponding to IRQ
//         The valid value range is: [16, 51], [54, 61], and 0xffff.
//         0xffff means enable all IRQs.
//
// RETURN:
// none
//
void EnIRQ(INT32U NVIC_No);

//
// FUNCTION:
// Disable IRQ
//
// PARAMETERS:
// NVIC_No - NVIC serial No. corresponding to IRQ
//         The valid value range is: [16, 51], [54, 61], and 0xffff.
//         0xffff means disable all IRQs.
//
// RETURN:
// none
//
void DisIRQ(INT32U NVIC_No);

//
// FUNCTION:
// Enable exception
//
// PARAMETERS:
// Exception - Which Exception to be controlled
//           Valid value include:
//           1. EXCEPTION_USAGEFAULT
//           2. EXCEPTION_BUSFAULT
//           3. EXCEPTION_MEMFAULT
//
// RETURN:
// none
//
void EnException(INT8U Exception);

//
// FUNCTION:
// Disable exception
```

```
//
// PARAMETERS:
// Exception - Which Exception to be controlled
//           Valid value include:
//           1. EXCEPTION_USAGEFAULT
//           2. EXCEPTION_BUSFAULT
//           3. EXCEPTION_MEMFAULT
//
// RETURN:
// none
//
void DisException(INT8U Exception);

//
// FUNCTION:
// Set IRQ
//
// PARAMETERS:
// NVIC_No - NVIC serial No. corresponding to IRQ
//         The valid value range is: [16, 51], [54, 61]
//
// RETURN:
// none
//
void SetIRQ(INT32U NVIC_No);

//
// FUNCTION:
// Clear IRQ
//
// PARAMETERS:
// NVIC_No - NVIC serial No. corresponding to IRQ
//         The valid value range is: [NVIC_FCS, NVIC_USB_2],
//         [NVIC_DMA_CH0, NVIC_DMA_CH7],0xffff means clear all IRQs.
//
// RETURN:
// none
//
void ClrIRQ(INT32U NVIC_No);

//
// FUNCTION:
// Configure IRQ priority group register (PRIGROUP)
//
// PARAMETERS:
// BitCnt - How many bits will be used to denote preempt priority. So
//        the left bits in Priority Level Register is used to denote sub-
//        priority.
//        The valid value range is [PREEMPT_BIT_CNT0, PREEMPT_BIT_CNT7].
//        Example: If BitCnt = 2, means bit7, bit6 are used to denote
//                preempt priority, and bit5~bit0 are used to denote
//                sub-priority.
//
// RETURN:
```

```
// none
//
void ConfPreemptPriority(INT8U BitCnt);

//
// FUNCTION:
// Configure IRQ priority
//
// PARAMETERS:
// NVIC_No - NVIC serial No. corresponding to IRQ
//         The valid value range is: [NVIC_FCS, NVIC_USB_2],
//         [NVIC_DMA_CH0, NVIC_DMA_CH7]
// PreemptFieldLen - how many bits are used as preempt priority bit
// Priority - IRQ priority, the valid value range is [PRI_LEVEL0,
//          PRI_LEVEL7]
//
// RETURN:
// none
//
void ConfPriorityForIRQ(INT32U NVIC_No, INT8U PreemptFieldLen, INT8U
Priority);

//
// FUNCTION:
// Configure Exception priority
//
// PARAMETERS:
// NVIC_No - NVIC serial No. corresponding to Exception
//         The valid value range is: [4, 6], [11, 12], [14, 15]
// PreemptFieldLen - how many bits are used as preempt priority bit
// Priority - Exception priority, the valid value range is [PRI_LEVEL0,
//          PRI_LEVEL7]
//
// RETURN:
// none
//
void ConfPriorityForException(INT32U NVIC_No, INT8U PreemptFieldLen,
INT8U Priority);

//
// FUNCTION:
// Restore interrupt enable register
//
// PARAMETERS:
// RegNo - serial No. of interrupt enable registers.
//       0 - SETENA0
//       1 - SETENA1
// Val32 - use this value to restore interrupt enable register according
to
//       RegNo
//
// RETURN:
// none
//
```

```
void RestoreIRQ (INT8U RegNo, INT32U Val32);

//
// FUNCTION:
// Enable exception
//
// PARAMETERS:
// Exception - Which Exception to be controlled
//           Valid value include:
//           1. EXCEPTION_USAGEFAULT
//           2. EXCEPTION_BUSFAULT
//           3. EXCEPTION_MEMFAULT
//
// RETURN:
// none
//
void EnException(INT8U Exception);
```

5.2.2 Functions in PeripheralCtrl folder

All functions in this folder are designed to drive in-chip macros.

As mentioned above, there are 2 sets of ways to access in-chip registers.

1) Function-Way – access registers by functions

2) Definition-Way – access registers by definitions

Here only to introduce the Function-Way, users can check the Definition-Way in *.h files in the folder of MCL/head/driver/PeripheralCtrl

These functions are in the file of MCL/source/driver/PeripheralCtrl.

- **Driver for clock**

```
//
// FUNCTION:
// Configure PLL clock frequency.
//
// PARAMETERS:
// CrystalFreq - External main crystal frequency(MHz)
// K_Divide - Factor of frequency division, valid value range is [0, 15].
// N_Multi - Factor of frequency multiplication, valid value range is [0,
// 31].
// MainClkWaitTime - How long to wait for the main clock stability.
//                Available value range is
//                [MAIN_CLK_WAITING_TIME0, MAIN_CLK_WAITING_TIME15]
// PLLClkWaitTime - How long to wait for the PLL stability.
//               Available value range is
//               [PLL_CLK_WAITING_TIME0, PLL_CLK_WAITING_TIME7]
//
// DESCRIPTION:
// The formula of calculating PLL output clock frequency is
// PLLClock = CrystalFreq * (N_Multi + 1) / (K_Divide + 1)
//
// RETURN:
// none
//
void ConfPLLClock(INT8U CrystalFreq, INT8U K_Divide, INT8U N_Multi, INT8U
MainClkWaitTime, INT8U PLLClkWaitTime);

//
// FUNCTION:
// Configure base clock frequency.
//
// PARAMETERS:
// Div - Factor of frequency division
//      Valid range is [BASE_CLK_DIV0, BASE_CLK_DIV6]
//
// RETURN:
```

```
// none
//
void ConfBaseClock(INT8U Div);


//
// FUNCTION:
// Configure APBx clock frequency.
//
// PARAMETERS:
// APBx - configure which APB
//       Valid range is [APB0, APB2]
// Div - Factor of frequency division
//      Valid range is [APBx_CLK_DIV0, APBx_CLK_DIV3]
//
// DESCRIPTION
// Each macro in-chip uses which APB bus clock is listed below.
// APB0 - Clock and Reset Control, Hardware Watch-dog Timer,
//       Software Watch-dog Timer, Dual Timer
// APB1 - Multi-function Timer unit0, Multi-function Timer unit1, PPG, Base
//       Timer.
// APB2 - Multi-function Serial, Watch Counter, USB Clock Generation
//       Circuit, CAN Pre-scaler.
//
// RETURN:
// none
//
void ConfAPBxClock(INT8U APBx, INT8U Div);
```

- **Driver for watchdog**

```
//
// FUNCTION:
// Configure software WDT clock.
//
// PARAMETERS:
// Div - Factor of frequency division
//      Valid range is [WDT_CLK_DIV0, WDT_CLK_DIV3]
//
// DESCRIPTION
// -----------------------------------
// Div         | description
// ------------|----------------------
// WDT_CLK_DIV0 | 1/1 [Initial value]
// WDT_CLK_DIV1 | 1/2
// WDT_CLK_DIV2 | 1/4
// WDT_CLK_DIV3 | 1/8
// -----------------------------------
//
// RETURN:
// none
//
void ConfWDTClock(INT8U Div);

//
// FUNCTION:
// configure WDT load register.
//
// PARAMETERS:
// WhichWDT - Which WDT to configure, valid value is HWDT or SWDT.
// ms - WDT overflow period (ms), valid range is from 0.001 to upper limit.
//
// RETURN:
// none
//
void ConfWDTLoadReg(INT8U WhichWDT, FP32 ms);

//
// FUNCTION:
// configure software WDT control register.
//
// PARAMETERS:
// WhichWDT - Which WDT to configure, valid value is HWDT or SWDT.
// RstEn - Reset by WDT is enabled or not.
//       Valid value is TRUE or FALSE.
// IntEn - Interrupt by WDT is enabled or not.
//       Valid value is TRUE or FALSE.
//
// DESCRIPTION
// Software WDT begins to run when set IntEn to TRUE.
//
```

```
// RETURN:
// none
//
void ConfWDTCtrlReg(INT8U WhichWDT, BOOL RstEn, BOOL IntEn);

//
// FUNCTION:
// Feed watch dog, clear WDT counter.
//
// PARAMETERS:
// WhichWDT - Which WDT to configure, valid value is HWDT or SWDT.
//
// DESCRIPTION
// Call this function will reload WDT load register.
//
// RETURN:
// none
//
void FeedWDT(INT8U WhichWDT);

//
// FUNCTION:
// Unlock registers relative to WDT.
//
// PARAMETERS:
// WhichWDT - Which WDT to configure, valid value is HWDT or SWDT.
//
// DESCRIPTION
// Call this function will reload WDT load register.
//
// RETURN:
// none
//
void INNER_FUNCTION UnlockWDT(INT8U WhichWDT);
```

- **Driver for GPIO**

```
//
// FUNCTION:
// Configure a pin act as peripheral function or not.
//
// PARAMETERS:
// WhichPort - Which port to be configured. Valid value is [PORT0, PORT7].
// WhichPin - Which pin to be configured. Valid value is [PIN0, PIN15].
// En - Peripheral function enable or not.
//
// RETURN:
// none
//
void ConfPinPeripheralFunc(INT8U WhichPort, INT8U WhichPin, BOOL En);

//
// FUNCTION:
// Configure a pin pull-up or not.
//
// PARAMETERS:
// WhichPort - Which port to be configured. Valid value is [PORT0, PORT7].
// WhichPin - Which pin to be configured. Valid value is [PIN0, PIN15].
// En - Pull-up enable or not.
//
// RETURN:
// none
//
void ConfPinPullUp(INT8U WhichPort, INT8U WhichPin, BOOL En);

//
// FUNCTION:
// Configure data direction for a pin.
//
// PARAMETERS:
// WhichPort - Which port to be configured. Valid value is [PORT0, PORT7].
// WhichPin - Which pin to be configured. Valid value is [PIN0, PIN15].
// Dir - Act as an input or output pin.
//      Valid value is IO_DIRECTION_IN. IO_DIRECTION_OUT.
//
// RETURN:
// none
//
void ConfPinDirection(INT8U WhichPort, INT8U WhichPin, INT8U Dir);

//
// FUNCTION:
// Read bit data from port.
//
// PARAMETERS:
// WhichPort - Which port to be configured. Valid value is [PORT0, PORT7].
// WhichPin - Which pin to be configured. Valid value is [PIN0, PIN15].
```

```
//
// RETURN:
// Read result.
//
BOOL ReadBitFromPort(INT8U WhichPort, INT8U WhichPin);

//
// FUNCTION:
// Read byte data from port.
//
// PARAMETERS:
// WhichPort - Which port to be configured. Valid value is [PORT0, PORT7].
//
// RETURN:
// Read result.
//
INT32U ReadByteFromPort(INT8U WhichPort);

//
// FUNCTION:
// Write bit data from port.
//
// PARAMETERS:
// WhichPort - Which port to be configured. Valid value is [PORT0, PORT7].
// WhichPin - Which pin to be configured. Valid value is [PIN0, PIN15].
// bVal - Bit value, valid value is HIGH_LEVEL, LOW_LEVEL.
//
// RETURN:
// none
//
void WriteBitToPort(INT8U WhichPort, INT8U WhichPin, BOOL bVal);

//
// FUNCTION:
// Write byte data from port.
//
// PARAMETERS:
// WhichPort - Which port to be configured. Valid value is [PORT0, PORT7].
// Val - Value
//
// RETURN:
// none
//
void WriteByteToPort(INT8U WhichPort, INT32U Val);
```

- **Driver for Dual Timer**

```
//
// FUNCTION:
// To enable dual timer.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
// RETURN:
// none
//
void EnDualTimer(INT8U WhichDualTimer);

//
// FUNCTION:
// To disable dual timer.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
// RETURN:
// none
//
void DisDualTimer(INT8U WhichDualTimer);

//
// FUNCTION:
// To configure dual timer work mode.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
// Mode - work mode. Valid value is DT_MODE_FREERUN, DT_MODE_PERIOD,
//        DT_MODE_ONESHOT.
//
// RETURN:
// none
//
void ConfDualTimerWorkMode(INT8U WhichDualTimer, INT8U Mode);

//
// FUNCTION:
// To enable dual timer interrupt.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
//
// RETURN:
// none
//
```

```
void EnDualTimerINT(INT8U WhichDualTimer);

//
// FUNCTION:
// To disable dual timer interrupt.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
//
// RETURN:
// none
//
void DisDualTimerINT(INT8U WhichDualTimer);

//
// FUNCTION:
// To clear dual timer interrupt.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
//
// RETURN:
// none
//
void ClrDualTimerINT(INT8U WhichDualTimer);

//
// FUNCTION:
// To read dual timer interrupt status.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
//
// RETURN:
// FALSE - No interrupt generated
// TRUE - Interrupt is generated
//
BOOL ReadIntStatusOfDualTimer(INT8U WhichDualTimer);

//
// FUNCTION:
// To prescale for dual timer interrupt.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
// PreVal - Prescale value. Valid value is DT_PRE_BY_1, DT_PRE_BY_16,
//          DT_PRE_BY_256
//
// RETURN:
// none
```

```
//
void PreOfDualTimer(INT8U WhichDualTimer, INT8U PreVal);

//
// FUNCTION:
// To read prescale of dual timer.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
//
// RETURN:
// Prescale value, 1, 16 or 256.
//
INT16S ReadPreOfDualTimer(INT8U WhichDualTimer);

//
// FUNCTION:
// To configure counter size for dual timer.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
// CntSize - Counter size. Valid value is DT_CNT_SIZE_16, DT_CNT_SIZE_32
//
// RETURN:
// none
//
void CntSizeOfDualTimer(INT8U WhichDualTimer, INT8U CntSize);

//
// FUNCTION:
// To configure reload value for dual timer.
// This function will configure dual timer's load register
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
// ReloadVal - reload value.
//
// RETURN:
// none
//
void ConfReloadValForDualTimer(INT8U WhichDualTimer, INT32U ReloadVal);

//
// FUNCTION:
// To configure reload value for dual timer.
// This function will configure dual timer's background load register
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
// ReloadVal - reload value.
```

```
//
// RETURN:
// none
//
void ConfBackReloadValForDualTimer(INT8U WhichDualTimer, INT32U
ReloadVal);

//
// FUNCTION:
// To read reload value from dual timer.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
//
// RETURN:
// Reload value
//
INT32U ReadReloadValFromDualTimer(INT8U WhichDualTimer);

//
// FUNCTION:
// To configure interrupt period for dual timer.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
// ms - Interrupt period(ms)
//
// RETURN:
// none
//
void ConfPeriodForDualTimer(INT8U WhichDualTimer, FP32 ms);

//
// FUNCTION:
// To configure interrupt period of dual timer for next time.
//
// PARAMETERS:
// WhichDualTimer - Which dual timer will be configured.
//                  valid value is DUAL_TIMER1, DUAL_TIMER2
// ms - Interrupt period(ms)
//
// RETURN:
// none
//
void ConfNextPeriodForDualTimer(INT8U WhichDualTimer, FP32 ms);
```

- **Driver for free run timer in MFT**

```
//
// FUNCTION:
// Configure count mode for FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1.
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
// Mode - Count mode, valid value is CNT_MODE_UP, CNT_MODE_UPDOWN.
//
// RETURN:
// none
//
void ConfCountModeForFRTx(INT8U WhichMFT, INT8U WhichFRTCh, INT8U Mode);

//
// FUNCTION:
// Configure count cycle for FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1.
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
// ms - count cycle(ms), valid range is from 0.001 to upper limit.
//
// RETURN:
// none
//
void ConfCntCycleForFRTx(INT8U WhichMFT, INT8U WhichFRTCh, FP32 ms);

//
// FUNCTION:
// Configure count cycle for FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1.
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
//
// RETURN:
// CNT_MODE_UP or CNT_MODE_UPDOWN
//
INT8U CheckCntModeForFRTx(INT8U WhichMFT, INT8U WhichFRTCh);

//
// FUNCTION:
// Start FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
```

```
// WhichFRTCh - Which FRT channel to configure.
//              Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
//
// RETURN:
// none
//
void StartFRTx(INT8U WhichMFT, INT8U WhichFRTCh);

//
// FUNCTION:
// Stop FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFRTCh - Which FRT channel to configure.
//              Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
//
// ATTENTION:
// There are two ways to use this register, as described below.
// 1. When starting FRT's counter operation:
//    When "0" is written to TCSA.STOP and TCSA.SCLR while FRT's count
//    operation is stopped, FRT starts counting.
// 2. When clearing the count value of FRT's counter to "0x0000" through
//    synchronization in Up-count mode:
//    If "0" is written to TCSA.STOP and "1" is written to TCSA.SCLR during
//    FRT's count operation in Up-count mode, FRT's count value is cleared
//    to "0x0000" in FRT's next count clock. If "0" is written to TCSA.SCLR
//    before the counter is cleared, the counter clear request is cancelled
//    and the counter value is not cleared. This operation cannot be
//    performed in Up/Down-count mode.
// 3. When stopping the operation of FRT's counter:
//    If "1" is written to TCSA.STOP and TCSA.SCLR during FRT's count
//    operation, FRT stops the count operation. In some cases, FRT's
//    counter value is not initialized to "0x0000" even after FRT stops,
//    depending on the state of FRT's count clock. Always write "0x0000"
to
//    TCDT afterwards to clear FRT's counter value to "0x0000".
//
// RETURN:
// none
//
void StopFRTx(INT8U WhichMFT, INT8U WhichFRTCh);

//
// FUNCTION:
// Enable or disable TCCP buffer for FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFRTCh - Which FRT channel to configure, valid value is CH0, CH1,
//              CH2.
// En - Enable buffer or not, valid value is TRUE, FALSE.
//
// RETURN:
```

```
// none
//
void ConfBufForFRTx(INT8U WhichMFT, INT8U WhichFRTCh, BOOL En);

//
// FUNCTION:
// Configure division value for FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFRTCh - Which FRT channel to configure.
//           Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
// Div - Division value
//      Valid value range is [FRT_DIV0, FRT_DIV8]
//
// DESCRIPTION:
// FRTs count clock = APB1 bus clock / 2^Div
//
// --------------------------------
// Div      | description
// ---------|----------------------
// FRT_DIV0 | 1/1 [Initial value]
// FRT_DIV1 | 1/2
// FRT_DIV2 | 1/4
// FRT_DIV3 | 1/8
// FRT_DIV4 | 1/16
// FRT_DIV5 | 1/32
// FRT_DIV6 | 1/64
// FRT_DIV7 | 1/128
// FRT_DIV8 | 1/256
// --------------------------------
//
// ATTENTION:
// Only valid in 16-bit up/down counter
//
// RETURN:
// none
//
void ConfDivFreqForFRTx(INT8U WhichMFT, INT8U WhichFRTCh, INT8U Div);

//
// FUNCTION:
// Configure division value for FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFRTCh - Which FRT channel to configure.
//           Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
//
// RETURN:
// [FRT_DIV0, FRT_DIV8]
//
INT8U ReadDivFreqForFRTx(INT8U WhichMFT, INT8U WhichFRTCh);
```

```
//
// FUNCTION:
// Enable or disable interrupt for FRTx when TCCP matches FRT count value.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
// En - Enable or not, valid value is TRUE, FALSE.
//
// RETURN:
// none
//
void ConfIntForFRT_MatchTop(INT8U WhichMFT, INT8U WhichFRTCh, INT8U En);

//
// FUNCTION:
// Clear interrupt flag for FRTx. Event: TCCP matches FRT count value.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
//
// RETURN:
// none
//
void ClrIntOfFRTx_MatchTop(INT8U WhichMFT, INT8U WhichFRTCh);

//
// FUNCTION:
// Enable or disable interrupt for FRTx when FRT count value reaches 0.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
// En - Enable or not, valid value is TRUE, FALSE.
//
// RETURN:
// none
//
void ConfIntForFRT_MatchZero(INT8U WhichMFT, INT8U WhichFRTCh, INT8U En);

//
// FUNCTION:
// Clear interrupt flag for FRTx of FRT count value reaches 0.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
//
// RETURN:
```

```
// none
//
void ClrIntOfFRTx_MatchZero(INT8U WhichMFT, INT8U WhichFRTCh);

//
// FUNCTION:
// Initialize FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
// En - Enable or not, valid value is TRUE, FALSE.
//
// ATTENTION:
// There are two ways to use this register, as described below.
// 1. When stopping FRT counter
//    Write "1" to issue an initialization request for FRT's operation
//    state when stopping FRT counter.
// 2. When clearing FRT counter through clock synchronization
//    Write "1" to issue a request to clear FRT's count value to "0x0000"
//    through synchronization when operating FRT in Up-count mode.
//
// RETURN:
// none
//
void InitFRTx(INT8U WhichMFT, INT8U WhichFRTCh, BOOL En);
```

- **Driver for output compare in MFT**

```
//
// FUNCTION:
// Configure each OC channel connect to which FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FTR_CH0, FTR_CH1, FTR_CH2.
//
// RETURN:
// none
//
void ConfFRTforOCChx(INT8U WhichMFT, INT8U WhichFRTCh, INT8U WhichOCCh);

//
// FUNCTION:
// Start OC channel to operate.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
//
// RETURN:
// none
//
void StartOCChx(INT8U WhichMFT, INT8U WhichOCCh);

//
// FUNCTION:
// Stop OC channel to operate.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
//
// RETURN:
// none
//
void StopOCChx(INT8U WhichMFT, INT8U WhichOCCh);

//
// FUNCTION:
// Enable or disable buffer for each OC channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
```

```
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
// En - Enable or disable, valid value is TRUE, FALSE.
//
// RETURN:
// none
//
void ConfBufForOCChx(INT8U WhichMFT, INT8U WhichOCCh, INT8U En);

//
// FUNCTION:
// Enable or disable interrupt for each OC channel when OCCP matches FRT
count
// value.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
// En - Enable or disable, valid value is TRUE, FALSE.
//
// RETURN:
// none
//
void ConfIntForOCChx_MatchFRT(INT8U WhichMFT, INT8U WhichOCCh, INT8U En);

//
// FUNCTION:
// Check interrupt is enabled or disabled for each OC channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
//
// RETURN:
// TRUE - enabled
// FALSE - disabled
//
BOOL ReadIntEnFlagWhenOCCPMatchFRT(INT8U WhichMFT, INT8U WhichOCCh);

//
// FUNCTION:
// Check match flag for OCx of OCCP matches FRT count.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
//
// RETURN:
// TRUE - matching detected
// FALSE - matching not detected
//
```

```
INT8U ReadIntFlag_OCCPMatchFRT(INT8U WhichMFT, INT8U WhichOCCh);

//
// FUNCTION:
// Clear match flag for OCx of OCCP matches FRT count.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
//
// RETURN:
// none
//
void ClrIntFlag_OCCPMatchFRT(INT8U WhichMFT, INT8U WhichOCCh);

//
// FUNCTION:
// Configure the initial output level for RTx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
// Level - initial output level, valid value is LEVEL_HIGH, LEVEL_LOW
//
// RETURN:
// none
//
void ConfInitOutputLevelForOCx(INT8U WhichMFT, INT8U WhichOCCh, INT8U
Level);

//
// FUNCTION:
// Configure the timing when data in OCCP buffer send to OCCP for RTx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
// When - Timing of send buffer, valid value is CNT_TOP, CNT_BOTTOM
//       CNT_TOP - Performs buffer transfer of the OCCP register upon Peak
//                 value detection by FRT.
//       CNT_BOTTOM - Performs buffer transfer of the OCCP register upon
//                    Peak value detection by FRT.
//
// RETURN:
// none
//
void ConfTimingOfSendBufForOCx(INT8U WhichMFT, INT8U WhichOCCh, INT8U
When);

//
// FUNCTION:
```

```
// Configure work mode for RTx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
// CMODE - see table below
// MOD - see table below
//
// RETURN:
// none
//
void ConfWorkModeForOCChx(INT8U WhichMFT, INT8U WhichOCCh, INT8U CMODE,
INT8U MOD);

//
// FUNCTION:
// Configure OCCP value for RTx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
// ms - Elapsed time from 0x0000 to the OCCP value which will be configured
//      in this function, unit is millisecond.
//
// DESCRIPTION:
// Below is the illustration.
//
//         TCCP
//          /\
//      OCCP/  \
//         /:   \
//        / :    \
//       /  :     \
//      /   :      \
//   -->: m :<--
//
// RETURN:
// none
//
void ConfOccpValForOCChxA(INT8U WhichMFT, INT8U WhichOCCh, FP32 ms);

//
// FUNCTION:
// Configure OCCP value for RTx corresponding to OC channelx.
//
// PARAMETERS:
// WhichMFT - Which MFT, valid value is MFT0, MFT1
// WhichOCCh - Which OC channel to configure.
//           Valid value range is [OC_CH0, OC_CH5].
// RegVal - Register value
//
// RETURN:
```

```
// none
//
void ConfOccpValForOCChxB(INT8U WhichMFT, INT8U WhichOCCh, INT16U RegVal);

//
// FUNCTION:
// Check RTx status.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// RTx - Check which RTx
//      Valid value is [OC_RT0, OC_RT5].
//
// RETURN:
// 1 - high level
// 0 - low level
//
INT8U ReadRTxForOCChx(INT8U WhichMFT, INT8U RTx);
```

- **Driver for waveform generator in MFT**

```
//
// FUNCTION:
// Start WFGxx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichWFGxx - Which WFGxx to configure.
//            Valid value is [WFG_CH10, WFG_CH54].
//
// RETURN:
// none
//
void StartWFGxx(INT8U WhichMFT, INT8U WhichWFGxx);

//
// FUNCTION:
// Stop WFGxx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichWFGxx - Which WFGxx to configure.
//            Valid value is [WFG_CH10, WFG_CH54].
//
// RETURN:
// none
//
void StoptWFGxx(INT8U WhichMFT, INT8U WhichWFGxx);

//
// FUNCTION:
// Start WFGxx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
//
// RETURN:
// none
//
void Start3PhaseWFG(INT8U WhichMFT);

//
// FUNCTION:
// Judge WFG is running or not.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichWFGxx - Which WFGxx to configure.
//            Valid value is [WFG_CH10, WFG_CH54].
//
// RETURN:
```

```
// FALSE - stopped
// TRUE - running
//
BOOL ReadWFGxxStatus(INT8U WhichMFT, INT8U WhichWFGxx);

//
// FUNCTION:
// Configure division value for WFGxx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichWFGxx - Which WFGxx to configure.
//           Valid value is [WFG_CH10, WFG_CH54].
// Div - Division value
//      Valid value range is [FRT_DIV0, FRT_DIV8]
//
// DESCRIPTION:
// This function configure WFG channel by pairs:
// (WFG_CH0, WFG_CH1),(WFG_CH2, WFG_CH3),(WFG_CH4, WFG_CH54).
//
// WFGs count clock = APB1 bus clock / 2^Div
// -------------------------------
// Div     | description
// --------|----------------------
// WFG_DIV0 | 1/1 [Initial value]
// WFG_DIV1 | 1/2
// WFG_DIV2 | 1/4
// WFG_DIV3 | 1/8
// WFG_DIV4 | 1/16
// WFG_DIV5 | 1/32
// WFG_DIV6 | 1/64
// -------------------------------
//
// RETURN:
// none
//
void ConfDivFreqForWFGxx(INT8U WhichMFT, INT8U WhichWFGxx, INT8U Div);

//
// FUNCTION:
// Configure work mode for WFGxx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichWFGxx - Which WFGxx to configure.
//           Valid value is [WFG_CH10, WFG_CH54].
// Mode - Division value
//      Valid value range is [MODE_THROUGH, MODE_RT-DEADTIMER] &
MODE_PPG-//      DEADTIMER
// GateOutput - What signal output from gate.
//           Valid value is [GATE_OUTPUT0, GATE_OUTPUT3]
//
// DESCRIPTION:
// This function configure WFG channel by pairs:
```

```
// (WFG_CH0, WFG_CH1),(WFG_CH2, WFG_CH3),(WFG_CH4, WFG_CH54).
//
// ----------------------------------------
// Mode              | description
// ------------------|---------------------
// MODE_THROUGH       | 0 [Initial value]
// MODE_RT_PPG        | 1
// MODE_TIME_PPG      | 2
// MODE_RT_DEAD_TIMER  | 4
// MODE_PPG_DEAD_TIMER | 7
// ----------------------------------------
//
// RETURN:
// none
//
void ConfWorkModeForWFGxx(INT8U WhichMFT, INT8U WhichWFGxx, INT8U Mode,
INT8U GateOutput);

//
// FUNCTION:
// Configure non-overlapped output signal polarity for WFGxx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichWFGxx - Which WFGxx to configure.
//            Valid value is [WFG_CH10, WFG_CH54].
// Polarity - Non-overlapped output signal polarity
//          Valid value is POLARITY_HIGH, POLARITY_LOW.
//
// DESCRIPTION:
// This function configure WFG channel by pairs:
// (WFG_CH0, WFG_CH1),(WFG_CH2, WFG_CH3),(WFG_CH4, WFG_CH54).
//
// RETURN:
// none
//
void ConfPolarityForWFGxx(INT8U WhichMFT, INT8U WhichWFGxx, INT8U
Polarity);

//
// FUNCTION:
// Configure WFG overflow period (ms) for WFGxx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichWFGxx - Which WFGxx to configure.
//            Valid value is [WFG_CH10, WFG_CH54].
// ms - WFG overflow period (ms), valid range is from 0.001 to upper limit.
//
// DESCRIPTION:
// This function configure WFG channel by pairs:
// (WFG_CH0, WFG_CH1),(WFG_CH2, WFG_CH3),(WFG_CH4, WFG_CH54).
//
// RETURN:
```

```
// none
//
void ConfCntValForWFGxx(INT8U WhichMFT, INT8U WhichWFGxx, FP32 ms);

//
// FUNCTION:
// Configure non-overlapped output signal polarity for WFGxx.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// En - Enable or not, valid value is TRUE, FALSE.
// NoiseTime - How long noise signal can be bear on DTTIX pin.
//           Valid value is [DTTI_NOISE_TIME0, DTTI_NOISE_TIME4]
//
// DESCRIPTION
// ------------------------------------------------
// DTTI_NOISE_TIME0 | 0 * PCLK (APB1 bus clock) |
// DTTI_NOISE_TIME1 | 4 * PCLK                 |
// DTTI_NOISE_TIME2 | 8 * PCLK                 |
// DTTI_NOISE_TIME3 | 16 * PCLK                |
// DTTI_NOISE_TIME4 | 32 * PCLK                |
// ------------------------------------------------
//
// RETURN:
// none
//
void ConfIntForWFG_DTTI(INT8U WhichMFT, BOOL En, INT8U NoiseTime);

//
// FUNCTION:
// Trigger a DTTI interrupt by software.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
//
// DESCRIPTION:
// This interrupt caused irrespective with bit NZCL.DTIE.
//
// RETURN:
// none
//
void TriggerDTTIBySoftware(INT8U WhichMFT);

//
// FUNCTION:
// Stop causing a DTTI interrupt by software.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
//
// DESCRIPTION:
// This function will clear WFIR.DTIF and clear the DTIF interrupt flag.
//
// RETURN:
```

```
// none
//
void StopDTTIBySoftware(INT8U WhichMFT);

//
// FUNCTION:
// Clear DTTI interrupt flag DTIF.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
//
// RETURN:
// none
//
void ClrDTIF(INT8U WhichMFT);

//
// FUNCTION:
// Judge WFGxx interrupt is generated or not.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichWFGxx - Which WFGxx to configure.
//            Valid value is [WFG_CH10, WFG_CH54].
//
// RETURN:
// FALSE - This interrupt is not generated
// TRUE - This interrupt is generated
//
BOOL CheckIntForWFGxx(INT8U WhichMFT, INT8U WhichWFGxx);

//
// FUNCTION:
// Clear WFGxx interrupt.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichWFGxx - Which WFGxx to configure.
//            Valid value is [WFG_CH10, WFG_CH54].
//
// RETURN:
// none
//
void ClrWFGIntStatusForWFGxx(INT8U WhichMFT, INT8U WhichWFGxx);

//
// FUNCTION:
// Select RTO pin for WFG and determine U/V/W is driven by which pin.
//
// PARAMETERS:
// WhichMFT - WFG is in Which MFT, valid value is MFT0, MFT1
// WhichPort, WhichPin - Which pin in which port will connect to RTO.
//                   WhichPort's valid value is [PORT0, PORT7]
//                   WhichPin's valid value is [PIN_0, PIN_15]
```

```
//
// RETURN:
// ChPairs - The pin described by WhichPort and WhichPin is in which
//         WFG channel pairs, valid value is WFG_CH10, WFG_CH32,
//         WFG_CH54
//
INT8S SelectOCCPCh(INT8U WhichMFT, INT8U WhichPort, INT8U WhichPin);

//
// FUNCTION:
// Configure pin work as RTO.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichPort, WhichPin - Which pin in which port will act as DTTI pin.
// ActMode - The mode of switching GPIO when DTTI comes
//         DTTI_SWITCH_GPIO_BY_HAND - switching GPIO by hand when
//                                    DTTIF interrupt arrives.
//         DTTI_SWITCH_GPIO_BY_AUTO - switching GPIO automatically when
//                                    DTTIF interrupt arrives.
//
// RETURN:
// none
//
void ConfPinActAsDTTI(INT8U WhichMFT, INT8U WhichPort, INT8S WhichPin,
INT8U ActMode);

//
// FUNCTION:
// Clear DTTI interrupt flag DTIF.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
//
// RETURN:
// Valid value is TRUE, FALSE.
//
BOOL ReadDTIF(INT8U WhichMFT);
```

- **Driver for A/D converter**

```
//
// FUNCTION:
// Enable or disable output convert ADC start signal when 0-detected INT
// comes.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFTRCh - Which FRT channel to configure.
//           Valid value is FRT_CH0, FRT_CH0, FRT_CH0.
// WhichADC - Start signal send to which ADC unit.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// En - Enable or not, valid value is TRUE, FALSE.
//
// RETURN:
// none
//
void ConfADCTriggerFromFRTx(INT8U WhichMFT, INT8U WhichFTRCh, INT8U
WhichADC, INT8U En);

//
// FUNCTION:
// Allocate channel to each ADC unit.
//
// PARAMETERS:
// ChSet - Which channels will allocate to ADC unit.
//        Every set bit in pChSet means to allocate a channel to ADC unit.
//        For example,
//        bit0=1 means allocate channel 0 to ADC unit
//        bit3=1 means allocate channel 3 to ADC unit
//        bit0=15 means allocate channel5 0 to ADC unit
// WhichADC - Allocate channel to which ADC unit
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// RETURN:
// none
//
void ConfADCChToUnit(INT32U ChSet, INT8U WhichADC);

//
// FUNCTION:
// Check scan conversion is finished or not.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// TRUE - Scan conversion is finished.
// FALSE - Scan conversion is not finished.
//
BOOL CheckFinishForScanConv(INT8U WhichADC);
```

```
//
// FUNCTION:
// Clear finish flag for scan conversion.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// none
//
void ClrIntFlagForScanConv(INT8U WhichADC);

//
// FUNCTION:
// Enable or disable interrupt when scan conversion is finished.
// When conversion values are written up to the stage count specified in
// SFNS, this bit is set to "1".
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// En - Enable or disable
//     Valid value is TRUE, FALSE.
//
// RETURN:
// none
//
void ConfConvFinishIntForScanConv(INT8U WhichADC, BOOL En);

//
// FUNCTION:
// Enable or disable interrupt when scan conversion is over-run.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// En - Enable or disable
//     Valid value is TRUE, FALSE.
//
// RETURN:
// none
//
void ConfOverRunIntForScanConv(INT8U WhichADC, BOOL En);

//
// FUNCTION:
// Stop ADC both scan and priority conversion.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
```

```
// RETURN:
// none
//
void StopADCConv(INT8U WhichADC);

//
// FUNCTION:
// Check scan conversion is stopped or not.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// TRUE - Scan conversion is stopped.
// FALSE - Scan conversion is not stopped.
//
BOOL CheckStopForScanConv(INT8U WhichADC);

//
// FUNCTION:
// Check data storage format for scan conversion.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// ADC_DATA_FORMAT_NORMAL - Converting result is stored at bit6~bit15 in
//                       SCFD.
// ADC_DATA_FORMAT_CHANGED - Converting result is stored at bit0~bit9 in
//                        SCFD.
//
BOOL CheckDataFormatForScanConv(INT8U WhichADC);

//
// FUNCTION:
// Check scan conversion FIFO is empty or not.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// TRUE - Scan conversion FIFO is empty.
// FALSE - Scan conversion FIFO is not empty.
//
BOOL CheckFIFOEmptyForScanConv(INT8U WhichADC);

//
// FUNCTION:
// Check scan conversion FIFO is full or not.
//
// PARAMETERS:
```

```
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// TRUE - Scan conversion FIFO is full.
// FALSE - Scan conversion FIFO is not full.
//
BOOL CheckFIFOFullForScanConv(INT8U WhichADC);

//
// FUNCTION:
// Check scan conversion FIFO is Overrun or not.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// TRUE - Scan conversion FIFO is Overrun.
// FALSE - Scan conversion FIFO is not Overrun.
//
BOOL CheckFIFOOverrunForScanConv(INT8U WhichADC);

//
// FUNCTION:
// Clear FIFO for scan conversion.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// none
//
void ClrFIFOForScanConv(INT8U WhichADC);

//
// FUNCTION:
// Enable or disable repeat when scan conversion is over-run.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// En - Enable or disable
//     Valid value is TRUE, FALSE.
//
// RETURN:
// none
//
void ConfRepeatForScanConv(INT8U WhichADC, BOOL En);

//
// FUNCTION:
// Start or restart(during conversion) ADC scan conversion.
```

```
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// none
//
void StartScanConv(INT8U WhichADC);

//
// FUNCTION:
// Configure FIFO stage.When A/D conversion data for the FIFO stage count
// set is written, the interrupt request flag (SCIF) is set to "1".
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// Stage - Valid range is [ADC_SCAN_CONV_STAGE1, ADC_SCAN_CONV_STAGE16]
//
// RETURN:
// none
//
void ConfFIFOStageForScanConv(INT8U WhichADC, INT8U Stage);

//
// FUNCTION:
// Configure FIFO stage.When A/D conversion data for the FIFO stage count
// set is written, the interrupt request flag (SCIF) is set to "1".
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// 31 - means no valid data can be read out from ADC FIFO.
// else - means a valid data read out from ADC FIFO.
//
INT32U ReadSingleData_ScanConv(INT8U WhichADC);

//
// FUNCTION:
// Select sampling time for an ADC channel.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be configured.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// WhichADCCh - Which ADC channel to be configured.
//            Valid value is [ADC_UNIT0, ADC_UNIT2].
// SampleTimeSel - Select sampling time.
//               Valid value is [ADC_SAMPLE_TIME0, ADC_SAMPLE_TIME1].
//
// RETURN:
// none
```

```
//
void SelectSampleTime(INT8U WhichADC, INT8U WhichADCCh, INT8U
SampleTimeSel);

//
// FUNCTION:
// Configure sampling time.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// WhichSampleTimeReg - Which sampling time register to be configured.
//                   Valid value is ADC_ST0, ADC_ST1.
// ST_Value - ST value in ADSTx register, valid value is [ADC_ST_VALUE0,
//          ADC_ST_VALUE63]
// STX_Value - STX value in ADSTx register, valid value is [ADC_STX0,
//           ADC_STX3].
//
// RETURN:
// none
//
void ConfSampleTime(INT8U WhichADC, INT8U WhichSampleTimeReg, INT8U
ST_Value, INT8U STX_Value);

//
// FUNCTION:
// Configure sampling time.
//
// PARAMETERS:
// WhichADC - Which ADC unit to be checked.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// CmpTime - Compare time, valid value is [ADC_CMP_TIME0, ADC_CMP_TIME7].
//
// RETURN:
// none
//
void ConfCmpTime(INT8U WhichADC, INT8U CmpTime);

//
//
// FUNCTION:
// ADC is triggered when ACCPx matches FRTy or disable triggered by ACCPx
and ACCPDNx.
// ACSA.SELx always equals 00, and other value is not supported temporarily.
// ACCPx buffer is not allowed.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFTRCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2, FRT_NONE.
// WhichADC - Start signal send to which ADC unit.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// ACCP_Value - ACCP value
//
```

```
// RETURN:
// none
//
void ConfTriggerByACCP(INT8U WhichMFT, INT8S WhichFRTCh, INT8U WhichADC,
INT16U ACCP_Value);

//
// FUNCTION:
// Configure trigger source for ADC when using scan conversion.
//
// PARAMETERS:
// WhichADC - Start signal send to which ADC unit.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
// TrigSrc - trigger source.
//         Valid value is TRG_SRC_NONE, TRG_SRC_FRT, TRG_SRC_BTC0, ...,
TRG_SRC_BTC7.
//
// RETURN:
// none
//
void ConfTrigSrcForScanConv(INT8U WhichADC, INT8U TrigSrc);

//
// FUNCTION:
// configure ADCMP channel connects to which FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1.
// WhichFTRCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
// WhichADCMP - Which ADCMP channel to be configured.
//            valid value is ADCMP_CH0, ADCMP_CH1, ADCMP_CH2
//
// RETURN:
// none
//
void ConfADCMPConnectToFRT(INT8U WhichMFT, INT8U WhichFRTCh, INT8U
WhichADCMP);

//
// FUNCTION:
// ADC is triggered when 0-detect occurs on FRTs.
// This function use the logic OR signal of FRTch.0 to ch.2 as selected
ADC
// trigger source.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1
// WhichFTRCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH0, FRT_CH0.
// WhichADC - Start signal send to which ADC unit.
//          Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
```

```
// none
//
void ConfADCTriggerBy0Detect_LogicOR(INT8U WhichMFT, INT8U WhichFRTCh,
INT8U WhichADC);

//
// FUNCTION:
// Configure a pin act as ADC analog input pin.
//
// PARAMETERS:
// WhichPort - Which port to be configured. Valid value is [PROT0, PROT7].
// WhichADCCh - Which ADC channel to be configured.
//            Valid value is [ADC_CH_0, ADC_CH_15].
//
// RETURN:
// none
//
void ConfPinUsedForADC(INT8U WhichPort, INT8S WhichADCCh);
```

5.3 Algorithm folder

5.3.1 Functions in Control Folder

These functions are in the folder of MCL/source/Algorithm/Control

```
//
// FUNCTION:
// According to the destination vector in fixed 2-axis system, program will
// send SVPWM waveform out.
//
// PARAMETERS:
// WhichMFT - Which MFT, valid value is MFT0, MFT1
// pstVoltageInFixAxis - the destination voltage vector in fixed axis
//
// RETURN:
// none
//
void    SendSVPWM_B(INT8U    WhichMFT,    volatile    _stDataInFixAxis
*pstVoltageInFixAxis);
```

5.3.2 Functions in Math Folder

These functions are in the folder of MCL/source/Algorithm/Math

```
//
// FUNCTION:
// calculate sin(theta)
//
// PARAMETERS:
// theta_Q15 - angle with Q15(32bit) format
//           Input any angle is enabled, because this function will
//           correct angle into a 0~360 degree.
//
// RETURN:
// value of sin(theta) with Q15(32bit) format
//
Q15_VAL32 Sin(Q15_VAL32 theta_Q15);

//
// FUNCTION:
// calculate cos(theta)
//
// PARAMETERS:
// theta_Q15 - angle with Q15(32bit) format
//           Input any angle is enabled, because this function will
//           correct angle into a 0~360 degree.
//
// RETURN:
// value of cos(theta) with Q15(32bit) format
//
Q15_VAL32 Cos(Q15_VAL32 theta_Q15);
```

5.3.3 Functions in Transfer Folder

These functions are in the folder of MCL/source/Algorithm/Transfer

```
//
// FUNCTION:
// Differential calculation by mathematical definition.
//
// PARAMETERS:
// DeltaVal_Q15 - Increment quantity of value
// DeltaT_Q15 - Increment quantity of time
//
// RETURN:
// Differential result.
//
Q15_VAL32 DiffDirect(Q15_VAL32 DeltaVal_Q15, Q15_VAL32 DeltaT_Q15);

//
// FUNCTION:
// Calculate the integral directly. If the calculation result exceed the
// maximum value(UpperLimit) or minimum value(LowerLimit), the maximum
// value(UpperLimit) or minimum value(LowerLimit) will be returned.
//
// PARAMETERS:
// x_Q15 - integral variable.
// t_Q15 - integral time (ms)
// UpperLimit - the upper limit of integral result.
// LowerLimit - the lower limit of integral result.
//
// RETURN:
// integral result.
//
Q15_VAL32 IntegralByDirect(Q15_VAL32 x_Q15, Q15_VAL32 t_Q15 /* ms */,
                      Q15_VAL32 UpperLimit, Q15_VAL32 LowerLimit);

//
// FUNCTION:
// First order LPF, this LPF is realized by an inertia function.
//
// PARAMETERS:
// pInfoOfLPF - relative information which is needed for LPF
//
// RETURN:
// Eqf, which is the BEMF output from LPF
//
Q15_VAL32 FirstOrderByInertia(volatile _stInfoOfLPF *pInfoOfLPF);

//
// FUNCTION:
// To calculate the output from PI regulator.
// PI method is to weak the integral component when the output(ui) exceeds
// the limit.
```

```
//
// PARAMETERS:
// pPI_Q15 - parameters used to calculate PI
//
// RETURN:
// output of PI regulator
//
Q15_VAL32 PI_WeakenI(volatile _stPIDPara *pPI_Q15);
```

5.3.4 Functions in Transformation Folder

These functions are in the folder of MCL/source/Algorithm/ Transformation

```
//
// FUNCTION:
// Clark transformation
//
// PARAMETERS:
// pstDataInThreeAxis - quantities in symmetrical 3-axis system
// pstDataInFixAxis - quantities in fixed 2-axis system
//
// RETURN:
// stDataInFixAxis - values transforming from symmetrical 3-axis system
//
void ClarkAmplitude(volatile _stDataInThreeAxis *pstDataInThreeAxis,
                volatile _stDataInFixAxis *pstDataInFixAxis);

//
// FUNCTION:
// Clark transformation
//
// PARAMETERS:
// pstDataInFixAxis - quantities in fixed 2-axis system
// pstDataInRotAxis - quantities in rotational 2-axis system
//
// RETURN:
// pstDataInRotAxis - values transforming from fixed 2-axis system
//
void Park(volatile _stDataInFixAxis *pstDataInFixAxis, volatile
_stDataInRotAxis *pstDataInRotAxis);

//
// FUNCTION:
// Park inverse transformation
//
// PARAMETERS:
// stDataInRotAxis - quantities in rotational 2-axis system
// stDataInFixAxis - quantities in fixed 2-axis system
//
// RETURN:
// stDataInFixAxis - values transforming from rotational 2-axis system
//
void InvPark(volatile _stDataInRotAxis *stDataInRotAxis, volatile
_stDataInFixAxis *stDataInFixAxis);

//
// FUNCTION:
// Correct angle to its main area([0, 360))
//
// PARAMETERS:
// theta_Q15 - angle with Q15(32bit) format
```

```
//
// RETURN:
// angle value in main area.
//
Q15_VAL32 CorrectAngleToMainArea(Q15_VAL32 theta_Q15);
```

5.4 Application folder

5.4.1 Functions in Init Folder

These functions are in the folder of MCL/source/Application/Init

```
//
// FUNCTION:
// To start dual timer
//
// WhichMFT - which dual timer will be operated.
//          Valid value is DUAL_TIMER1, DUAL_TIMER2.
//
// RETURN:
// none
//
void StartMacro_DT(INT8U WhichDualTimer);
//
// FUNCTION:
// To stop dual timer
//
// WhichMFT - which dual timer will be operated.
//          Valid value is DUAL_TIMER1, DUAL_TIMER2.
//
// RETURN:
// none
//
void StopMacro_DT(INT8U WhichDualTimer);
//
// FUNCTION:
// To start FRT
//
// WhichMFT - The FRT in which MFT will be operated.
//          Valid value is MFT0, MFT1.
//
// RETURN:
// none
//
void StartMacro_FRT(INT8U WhichMFT);

//
// FUNCTION:
// To stop FRT
//
// WhichMFT - The FRT in which MFT will be operated.
//          Valid value is MFT0, MFT1.
//
// RETURN:
// none
//
void StopMacro_FRT(INT8U WhichMFT);
```

```
// FUNCTION:
// To start OC
//
// PARAMETERS:
// WhichMFT - The OCU in which MFT will be operated.
//          Valid value is MFT0, MFT1.
//
// RETURN:
// none
//
void StartMacro_OC(INT8U WhichMFT);

// FUNCTION:
// To stop OC
//
// PARAMETERS:
// WhichMFT - The OCU in which MFT will be operated.
//          Valid value is MFT0, MFT1.
//
// RETURN:
// none
//
void StopMacro_OC(INT8U WhichMFT);

// FUNCTION:
// To start ADC
//
// PARAMETERS:
// none
//
// RETURN:
// none
//
void StartMacro_ADC(void);

// FUNCTION:
// To stop ADC
//
// PARAMETERS:
// none
//
// RETURN:
// none
//
void StopMacro_ADC(void);

//
// FUNCTION:
// To start macros, include:
// 1) ADC
// 2) OC
// 3) FRT
// 4) DualTimer
//
```

```
// PARAMETERS:
// WhichMFT - The motor controlled by this MFT will be stopped.
//          Valid value is MFT0, MFT1.
//
// RETURN:
// none
//
void StartMacros_MCL(INT8U WhichMFT);

//
// FUNCTION:
// To stop macros, include:
// 1) ADC
// 2) OC
// 3) FRT
// 4) DualTimer
//
// PARAMETERS:
// WhichMFT - The motor controlled by this MFT will be stopped.
//          Valid value is MFT0, MFT1.
//
// RETURN:
// none
//
void StopMacros_MCL(INT8U WhichMFT);
```

5.4.2 Functions in MotorCtrl Folder

These functions are in the folder of MCL/source/Application/MotorCtrl

```
//
// FUNCTION:
// Initialization.
// Do the following jobs:
// 1) initialize some global parameters
// 2) initialize NVIC
// 3) initialize clock
// 4) initialize WDT
// 5) initialize MFT
// 6) initialize ADC
// 7) initialize other global variables according to the user given value
//
// PARAMETERS:
// none.
//
// RETURN:
// 1  - successful
// -1 - failed
//
INT8S AutoConfByMCL(void);
//
// FUNCTION:
// To configure INT priority for NVIC.
//
// PARAMETERS:
// none
//
// RETURN:
// none
//
void ConfNVIC(void);
//
// FUNCTION:
// To configure PLL clock and bus clock.
//
// PARAMETERS:
// none
//
// RETURN:
// none
//
void ConfClock(void);

//
// FUNCTION:
// To configure GPIO port.
//
// PARAMETERS:
// none
//
```

```
// RETURN:
// none
//
void ConfGPIO(void);

//
// FUNCTION:
// To configure WDT.
//
// PARAMETERS:
// none
//
// RETURN:
// none
//
void ConfWDT(void);

//
// FUNCTION:
// To configure dual timer2.
//
// PARAMETERS:
// none
//
// RETURN:
// none
//
void ConfDualTimer2(void);

//
// FUNCTION:
// To configure MCU parameters
//
// PARAMETERS:
// none
//
// RETURN:
// none
//
void ConfMCUPara(void);

//
// FUNCTION:
// To configure motor parameters
//
// PARAMETERS:
// WhichMFT - Which MFT, valid value is MFT0, MFT1
//
// RETURN:
// none
//
void ConfMotorPara(INT8U WhichMFT);

//
```

```
// FUNCTION:
// To configure system parameters
//
// PARAMETERS:
// WhichMFT - Which MFT, valid value is MFT0, MFT1
//
// RETURN:
// none
//
void ConfSysPara(INT8U WhichMFT);

//
// FUNCTION:
// To configure control parameters
//
// PARAMETERS:
// WhichMFT - Which MFT, valid value is MFT0, MFT1
//
// RETURN:
// none
//
void ConfCtrlPara(INT8U WhichMFT);

//
// FUNCTION:
// To configure FRT
//
// PARAMETERS:
// WhichMFT - Which MFT, valid value is MFT0, MFT1
//
// RETURN:
// none
//
void ConfFRT(INT8S WhichMFT);

//
// FUNCTION:
// To configure OCCP
//
// PARAMETERS:
// WhichMFT - Which MFT, valid value is MFT0, MFT1
//
// RETURN:
// none
//
void ConfOCCP(INT8S WhichMFT);

//
// FUNCTION:
// To configure WFG
//
// PARAMETERS:
// WhichMFT - Which MFT, valid value is MFT0, MFT1
//
```

```
// RETURN:
// none
//
void ConfWFG(INT8S WhichMFT);

//
// FUNCTION:
// To configure ADC
//
// PARAMETERS:
// none
//
// RETURN:
// none
//
void ConfADC(void);

//
// FUNCTION:
// To configure DTTI
//
// PARAMETERS:
// WhichMFT - Which MFT, valid value is MFT0, MFT1
//
// RETURN:
// none
//
void ConfDTTI(INT8S WhichMFT);
//
// FUNCTION:
// Check flag to run or stop motor.
//
// PARAMETERS:
// WhichMFT - The motor controlled by this MFT will be operated.
//          Valid value is MFT0, MFT1.
//
// RETURN:
// none
//
void CtrlMotor(INT8U WhichMFT);

//
// FUNCTION:
// To orient rotor to a given position.
//
// PARAMETERS:
// WhichMFT - The motor controlled by this MFT will be stopped.
//          Valid value is MFT0, MFT1.
// VecAmp_Q15 - vector amplitude
// RotorDir_Q15 - position which rotor will orient to.
// SendPeriod - how long the vector will be last (ms)
//
// RETURN:
// none
```

```
//
void RotorOrienting(INT8S WhichMFT, Q15_VAL32 VecAmp_Q15,
                    Q15_VAL32 RotorDir_Q15, INT16U SendPeriod);
```

5.4.3 Functions in Others Folder

These functions are in the folder of MCL/source/Application/Others

```
//
// FUNCTION:
// Feed watch dog
//
// PARAMETERS:
// EnHWDT - Whether feed hardware watch dog or not.
// EnSWDT - Whether feed software watch dog or not.
//
// RETURN:
// none
//
void FeedDog(BOOL EnHWDT, BOOL EnSWDT);
//
// FUNCTION:
// Delay by software.
//
// PARAMETERS:
// DelayTime - how long is to delay (ms).
//
// RETURN:
// none
//
void DelaySoft(FP32 DelayTime);
//
// FUNCTION:
// After system powered on, the relay will be shorted.
//
// PARAMETERS:
// WhichPort - which port is to control the relay.
// WhichPin - which pin is to control the relay.
// ActiveLevel - The active levle for driving relay, valid value is
//             LOW_LEVEL, HIGH_LEVEL
// DelayTime - how long is to delay after power on, after the delay, relay
//           will be shorted (ms).
//
// RETURN:
// none
//
void ShortRelay(INT8U WhichPort, INT8U WhichPin, INT8U ActiveLevel, FP32
DelayTime);
```

6 Global Variables

Name	gstWarningCode
Type	struct _stWarningCode
Description	To record warning code, the warning amount of this variable can hold MAX_WARNING_CODE_CNT.

Name	gstErrInfo
Type	struct _stErrInfo
Description	To record error information. System will stop running the motor when some errors are occurred. *This variable is available when DEBUG_MODE is TRUE.*

Name	gADCSampleTime[][]
Type	FP32
Description	To save ADC sampling time, the unit is us (micro-second).

Name	gADCCmpTime[]
Type	FP32
Description	To save ADC compare time, the unit is us (micro-second).

Name	gFreqOfPLL
Type	FP32
Description	To save PLL output frequency, the unit is MHz.

Name	gFreqOfBaseClk
Type	FP32
Description	To save base clock frequency, theunit is MHz.

Name	gFreqOfAPB0Clk
Type	FP32
Description	To save APB0 clock frequency, the unit is MHz.

Name	gFreqOfAPB1Clk
Type	FP32
Description	To save APB1 clock frequency, the unit is MHz.

Name	gFreqOfAPB2Clk
Type	FP32
Description	To save APB2 clock frequency, the unit is MHz.

Name	gCntTopValOfFRT[][]
Type	INT32S
Description	To save the count top value for FRT, that is TCCP value.

Name	gPeriodOfFRT_Q15[][]
Type	Q15_VAL32
Description	To save the count top value for FRT, that is TCCP value.

Name	gFreqOfFRT[][]
Type	FP32
Description	To save the FRT frequency, unit is MHz.

Name	gFreqOfWFGCounter[][]
Type	FP32
Description	To save the WFG counter frequency, the unit is MHz.

Name	gFreqOfWDTClk
Type	FP32
Description	To save the WDT frequency, the unit is MHz.

Name	gstSapmle
Type	_stSapmle
Description	ADC properties and sampling results are stored.

Name	_stVectorA
Type	FP32
Description	To save the WDT frequency, the unit is MHz.

Name	gstMcuPara
Type	_stMcuPara
Description	To save the MCU parameters given by user.

Name	gstMotorPara[]
Type	_stMotorPara
Description	To save the motor parameters given by user.

Name	gstLPFPara[]
Type	_stLPFPara
Description	To save the LPF parameters given by user.

Name	gstDCBusPara[]
Type	_stDCBusPara
Description	To save the DC Bus parameters given by user.

Name	gstPIDPara[]
Type	_stPIDPara
Description	To save the PID parameters given by user.

Name	gstTrgSrcForAlgo[]
Type	_stTrgSrcForAlgo
Description	MFT uses a FRT channel to trigger running of motor control algorithm

Name	guniStartFlagForMFT[]
Type	_uniStartFlagForMFT
Description	A flag to define a FRT channel to count selected MFT.

Name	guniStartFlagForADC
Type	_uniStartFlagForADC
Description	A flag to define ADC unit.

Name	gstPinToDrvUVW[]
Type	_stPinToDrvUVW
Description	Save driving pin assignment for phase U/V/W

Name	gstDualTimer2
Type	_stDualTimer2
Description	dual timer-ch2 property

Name	gstMotorRunCtrl[]
Type	_stMotorRunCtrl
Description	Save parameters related to drive motor gstMotorRunCtrl[].Givens – Save the value given by user gstMotorRunCtrl[].Flags – system running status gstMotorRunCtrl[].Limit – system limit parameters when motor running gstMotorRunCtrl[].Cnts – define the control timing gstMotorRunCtrl[].ObsForPLL – save result of PLL algorithm gstMotorRunCtrl[].Derives – derivation quantities according to the user input and running status gstMotorRunCtrl[].PI – PI regulator parameters and PI input / output gstMotorRunCtrl[].Output – record some output quantities.

Name	math_sin_tbl[]
Type	INT16U
Description	Sine table.

7 User Interface I

Users maybe want to control motors by using MCL motor control algorithm directly and simply. This section introduces how to build such system.

7.1 Macros on-Chip Drive Interface

MCL has 2 types of macro driver interface. One is realized by definition in header files (*.h), the other is realized by functions in source files (*.c). The 2 types of interface can be found in the folder of MCL/head/drive/.

7.1.1 Interface of Definition Type

This interface offers approaching to access bits in registers.
To be easy that users contrast definitions with chip user manual, every definition is named by the bit name. For example, to access MODE bit of TCSAy in MFTx, MCL uses the following definition.

```
#define SET_MFTx_TCSAy_MODE(WhichMFT, WhichFRTCh, Mode) \
    do \
    { \
        ASSERT((CNT_MODE_UP == Mode) || (CNT_MODE_UPDOWN == Mode)); \
        \
        JOIN5(MFT, WhichMFT, _TCSA, WhichFRTCh, _bit).MODE = Mode; \
    }\
    while(FALSE);
```

To be easy using for user, every definition also has an alias to denote its function. For example, to configure the counter mode for FRT, MCL uses the following definition.

```
#define ConfCountModeOfFRTx_Def(WhichMFT, WhichFRTCh, Mode) \
        SET_MFTx_TCSAy_MODE(WhichMFT, WhichFRTCh, Mode)
```

If users want to set the counter mode of FRT0 in MFT0 to up-down counter mode, the following coding can be used.

```
ConfCountModeOfFRTx_Def(MFT0, FRT0, CNT_MODE_UPDOWN);
```
or
```
SET_MFTx_TCSAy_MODE(MFT0, FRT0, CNT_MODE_UPDOWN);
```

To see more definition type interface, please refer to the files of MCL/head/drive/*.h.

7.1.2 Interface of Function Type

There is an example to explain this interface. The interface is designed as follows

```
//
// FUNCTION:
// Configure count mode for FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1.
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
// Mode - Count mode, valid value is CNT_MODE_UP, CNT_MODE_UPDOWN.
//
// RETURN:
// none
//
void ConfCountModeForFRTx(INT8U WhichMFT, INT8U WhichFRTCh, INT8U Mode);
```

If users want to set the counter mode of FRT0 in MFT0 to up-down counter mode, the following coding can be used.

```
ConfCountModeForFRTx(MFT0, FRT0, CNT_MODE_UPDOWN);
```

Further, this interface can not only configure bits in registers but also do some complex jobs. For example, the function as follows can configure several registers automatically to meet advance demanding.

```
//
// FUNCTION:
// Configure count cycle for FRT channel.
//
// PARAMETERS:
// WhichMFT - Which MFT will be configured, valid value is MFT0, MFT1.
// WhichFRTCh - Which FRT channel to configure.
//            Valid value is FRT_CH0, FRT_CH1, FRT_CH2.
// ms - count cycle(ms), valid range is from 0.001 to upper limit.
//
// RETURN:
// none
//
void ConfCntCycleForFRTx(INT8U WhichMFT, INT8U WhichFRTCh, FP32 ms);
```

If users want to configure FRT counter period to 0.2ms, the following coding can be used.

```
ConfCntCycleForFRTx(MFT0, FRT0, 0.2);
```

To see more definition type interface, please refer to the files of MCL/head/drive/*.h.

7.2 User Parameters configuration Interface

7.2.1 Introduction

User parameters interface is used to accept parameters derived from hardware design. The interface has 2 parts: data structure interface and function interface. Users can modify the file of MCL / head / application / MotorCtrl / UserInterface_MCL.c to set new parameters. Below table describes the details.

Table 2 User Parameters Interface

Data Structure	Function	Description
_stParaForNVIC	ConfNVIC ()	Configure exception and INT priority
_stParaForMotor	ConfMotorPara()	Configure motor parameters
_stParaForDCBus	ConfSysPara()	Configure system parameters
_stParaForMcu	ConfMCUPara()	Configure MCU parameters
_stParaForClock	ConfClock ()	Configure PLL and bus clock
--	ConfWDT ()	Configure WDT
_stPortPinForGPIO	ConfGPIO()	Configure GPIO
_stParaForDualTimer2	ConfDualTimer2 ()	Configure dual timer-ch2
_stParaForMotorRunCtrl	ConfCtrlPara ()	Configure motor control parameters
_stParaForPID		Configure PID regulator parameters
_stParaForLPF	InitLPF_MCL()	Configure LPF
_stInitFRT	ConfFRT ()	Configure FRT
_stInitOCCP	ConfOCCP ()	Configure OCCP
_stInitWFG	ConfWFG ()	Configure WFG
_stInitADC	ConfADC ()	Configure ADC
_stInitDTTI	ConfDTTI ()	Configure DTTI

There is just an example to show the interface.

- Configure motor parameters

```
void ConfMotorPara(INT8U WhichMFT)
{
    _stParaForMotor stMotorPara;

    stMotorPara.PolePairs = 2;
    stMotorPara.RsPhase = 0.91f / 2;
    stMotorPara.Ld = 0.0107f;
    stMotorPara.Lq = 0.0119f;
    stMotorPara.K_phi = 36.2f; // Vrms/Krpm

    InitMotorPara_MCL(&stMotorPara, &gstMotorPara[WhichMFT]);
}
```

Figure 1 example for configure motor parameters

Referring the figure 3, the users will feel that it is easy to set the motor parameters into _stParaForMotor, and then call InitMotorPara_MCL() to deliver parameters into MCL.

7.2.2 How to Use

Users should modify the given data in the structure to suit for the hardware or system.

For example, if the following configuration is one of user's demanding configurations:

1. Configure for FRT0 in MFT0
2. Trigger period of MCL motor control algorithm is 200us
3. FRT counter mode is up-down count
4. Disable FRT buffer
5. Disable interruption when FRT counter becomes top value in the up-down count.
6. Enable interruption when FRT counter becomes zero.
7. FRT acts as trigger source for MCL motor control algorithm

- Step 1

  Referring Table 2, you can see that _stInitFRT and ConfFRT () should be used. At first, open the file of MCL / head / application / MotorCtrl / UserInterface_MCL.c, and then, locate to the function of ConfFRT (). _stInitFRT should be configured as follows:

Table 2 mainly configuration contents

Configuration	Description
stInitFRT.stConfFRT.WhichMFT = WhichMFT;	To meet the demanding 1. WhichMFT is determined by the input parameter of ConfFRT ()
stInitFRT.stConfFRT.WhichFRTCh = FRT_CH0;	To meet the demanding 1.
stInitFRT.stConfFRT.Cycle = 0.2;	To meet the demanding 2.
stInitFRT.stConfFRT.CntMode = CNT_MODE_UPDOWN;	To meet the demanding 3.
stInitFRT.stConfFRT.EnBuf = FALSE;	To meet the demanding 4.
stInitFRT.stConfFRT.EnINT_MatchTop = FALSE;	To meet the demanding 5.
stInitFRT.stConfFRT.EnINT_MatchZero = TRUE;	To meet the demanding 6.
stInitFRT.stConfFRT.TrgSrcForAlgo = TRUE;	To meet the demanding 7.

- Step 2

  Locating to the function of AutoConfByMCL() , input parameters in ConfFRT() should be given to determine which MFT is to be configured. As shown in the following figure.

```
INT8S AutoConfByMCL(void)
{
    ConfNVIC();
    ConfMCUPara();
    ConfMotorPara(MFT0);
    ConfSysPara(MFT0);
    ConfClock();
    ConfWDT();
    ConfGPIO();
    ConfDualTimer2();
    ConfCtrlPara(MFT0);
    ConfFRT(MFT0);
    ConfOCCP(MFT0);
    ConfWFG(MFT0);
    ConfADC(ADC_UNIT0);
    ConfDTTI(MFT0);

    return 1;
}
```

Figure 2 Setting input parameters for ConfFRT()

- Step 3

  Repeat step 1 and step 2 until completed all the configurations.

  Then, call AutoConfByMCL() in main() to start configuration progress, user configuration will be set.

```
#define DEFINE_GLOBAL_VARS
#include "User\Head\HeadInOne_User.h"

void main(void)
{
    //
    // Initialize all in-chip resources
    //
    AutoConfByMCL();
```

Figure 3 Calling AutoConfByMCL() in main()

7.3 Motor Driving interface

CtrlMotor() is the motor driving interface. After configuring the user parameters, as shown in section 7.2.2, Call this function in main loop and it can trigger the motor control algorithm in MCL.

```
void main(void)
{
    //
    // Initialize all in-chip resources
    //
    AutoConfByMCL();

    //
    // short the relay
    //
    ShortRelay(PORT2, PIN_1, TRUE, 1500);

    //************************************
    //
    // main loop start from here
    //
    //************************************

    while(TRUE)
    {
        FeedDog(TRUE, FALSE);

        CtrlMotor(MFTO);
    }
}
```

Figure 4 Calling CtrlMotor() to trigger motor control algorithm in MCL

8 User Interface II

Users maybe want to control motors by themselves algorithm. This section introduces how to use component in MCL.

8.1 Macros on-Chip Drive Interface

Please refer 7.1

8.2 User Parameters configuration Interface

Please refer 7.2

8.3 Macros on-chip Startup Interface

After configuring the user parameters, as shown in section 7.2.2, the following functions can be called to startup macros.

Function	Description
void StartMacro_DT(INT8U WhichDualTimer)	Startup dual timer
void StartMacro_FRT(INT8U WhichMFT)	Startup FRT
void StartMacro_OC(INT8U WhichMFT)	Startup OCCP
void StartMacro_ADC(void)	Startup ADC

8.4 Macros In-chip Stop Interface

The following functions can be called to stop macros.

Function	Description
void StopMacro_DT(INT8U WhichDualTimer)	Stop dual timer
void StopMacro_FRT(INT8U WhichMFT)	Stop FRT
void StopMacro_OC(INT8U WhichMFT)	Stop OCCP
void StopMacro_ADC(void)	Stop ADC

8.5 Log Warning Interface

Function	Description
void LogWarningCode(const INT16U WarningCode)	Record WarningCode into structure of _stWarningCode

8.6 Math interface

Function	Description
Q15_VAL32 Cos(Q15_VAL32 theta_Q15)	Calculating for cos
Q15_VAL32 Sin(Q15_VAL32 theta_Q15)	Calculating for sin

8.7 Transfer function interface

Function	Description
Q15_VAL32 DiffDirect(Q15_VAL32 DeltaVal_Q15, Q15_VAL32 DeltaT_Q15)	Calculating for differential
Q15_VAL32 IntegralByDirect(Q15_VAL32 x_Q15, Q15_VAL32 t_Q15 , Q15_VAL32 UpperLimit, Q15_VAL32 LowerLimit);	Calculating for integral
Q15_VAL32 FirstOrderByInertia(volatile _stInfoOfLPF *pInfoOfLPF)	Fist order LPF
Q15_VAL32 PI_WeakenI(volatile _stPIDPara *pPI_Q15)	PI regulator

8.8 Transformation function interface

Function	Description
void ClarkAmplitude(volatile _stDataInThreeAxis *pstDataInThreeAxis, volatile _stDataInFixAxis *pstDataInFixAxis)	Clark transformation
void Park(volatile _stDataInFixAxis *pstDataInFixAxis, volatile _stDataInRotAxis *pstDataInRotAxis)	Park transformation
void InvPark(volatile _stDataInRotAxis *stDataInRotAxis, volatile _stDataInFixAxis *stDataInFixAxis)	Inverting park transformation
Q15_VAL32 CorrectAngleToMainArea(Q15_VAL32, theta_Q15)	Correct angle to its main area – (0, 360)

8.9 Motor Control Algorithm Interface

Before calling these functions, please confirm that the user parameters are correctly set, as shown in section 7.2.2.

Function	Description
void SendSVPWM_B(INT8U WhichMFT, volatile _stDataInFixAxis *pstVoltageInFixAxis)	Send SVPWM vector
void RotorOrienting(INT8S WhichMFT, Q15_VAL32 VecAmp_Q15, Q15_VAL32 RotorDir_Q15, INT16U SendPeriod)	Orienting rotor to the given direction
Q15_VAL32 CalcAngleIncrement(INT8U WhichMFT, Q15_VAL32 NextSpeed_Q15)	Calculating the angle increment when motor runs.

8.10 Others

Function	Description
void DelaySoft(FP32 DelayTime)	Soft delay
void FeedDog(BOOL EnHWDT, BOOL EnSWDT)	Clear watch dog counter
void ShortRelay(INT8U WhichPort, INT8U WhichPin, INT8U ActiveLevel, FP32 DelayTime)	Short relay after delaying sometime

8.11 Definition Interface

Function	Description
Qn_Qm(Qn, n, m)	Transform a Qn value to Qm value

9 Installation and Usage for MCL

9.1 Installation

MCL can be installed from a compress package, as shown in the following figure. Because of the different version, the installation package maybe has a different name.

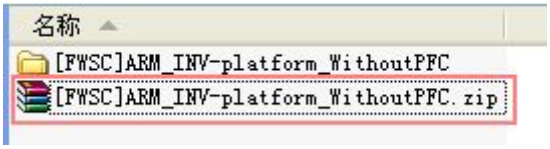


Figure 5 MCL installation package

If users have got this installation package, please refer the next step.

1. Unpack installation package to any folder.

   Unpack it and put it on the desktop.

Figure 6 unpack installation package on the desktop

2. Change the folder name something you want.

   Rename it to ArmInvPrj, for example.

Figure 7 Rename folder

3. Enter the folder of EWARM project: ArmInvPrj \ editor \ EWARM, and then open project file ArmInverterPlatform.eww.

Figure 8 EWARM project file

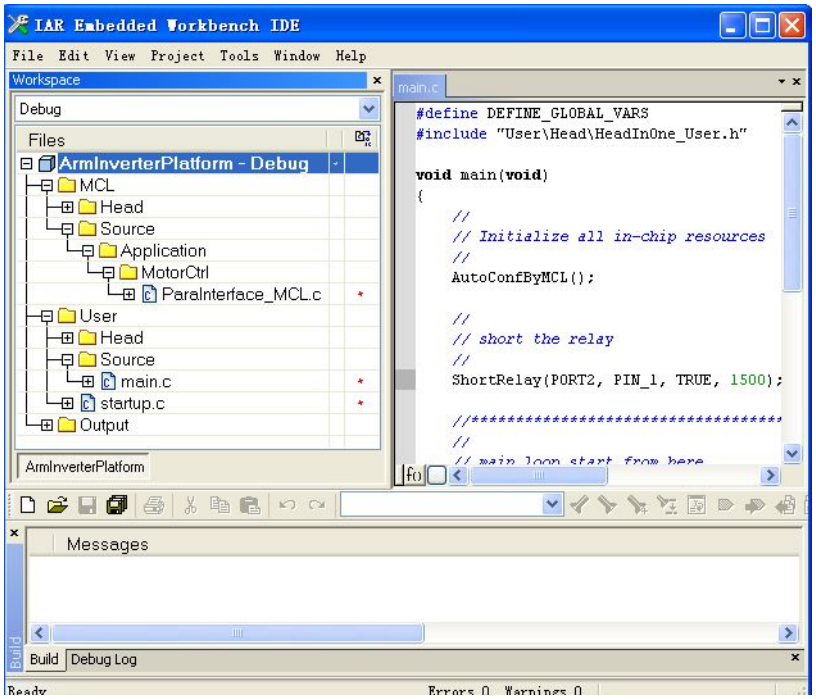


Figure 9 open project file

4. Right click on the project and choose rebuild all.

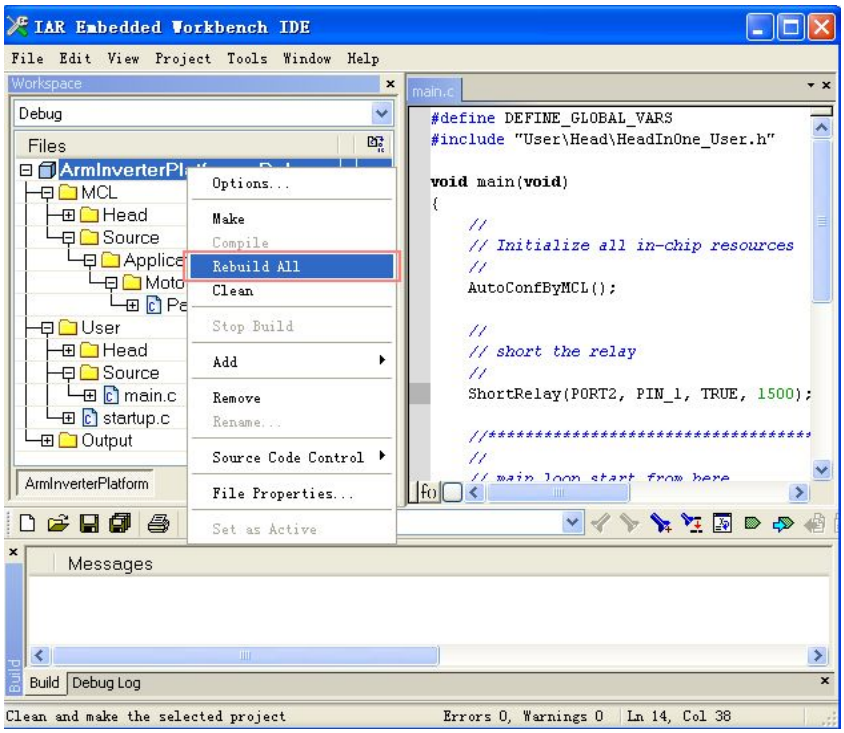


Figure 10 rebuild project

5. If no any error and warning message in the build box, it means the MCL is compiled correctly.

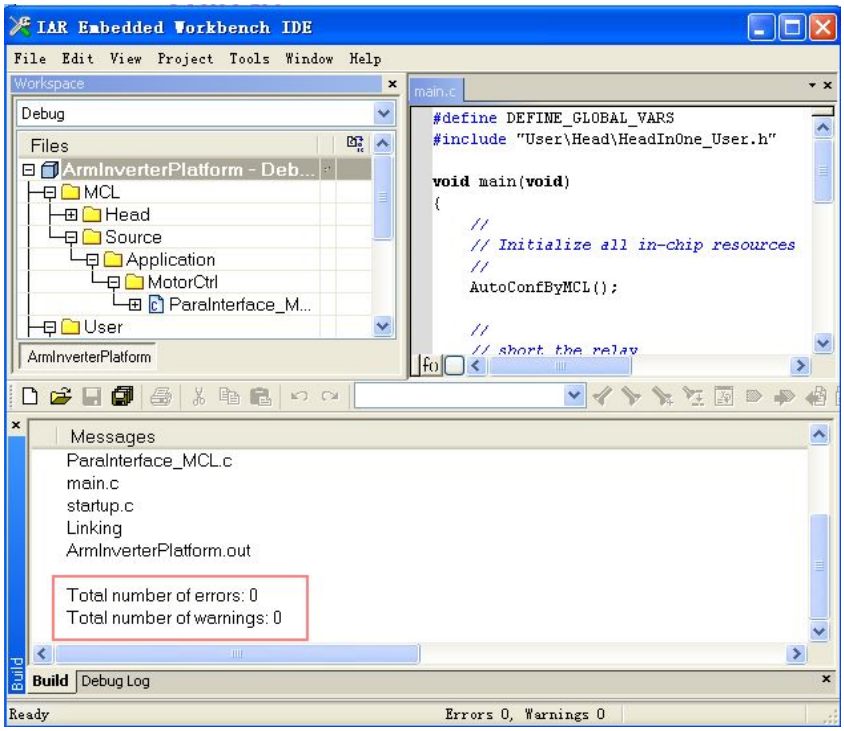


Figure 11 MCL is correctly compiled

6. If you meet some errors or warnings when rebuilding, please assure the following configuration in EWARM.

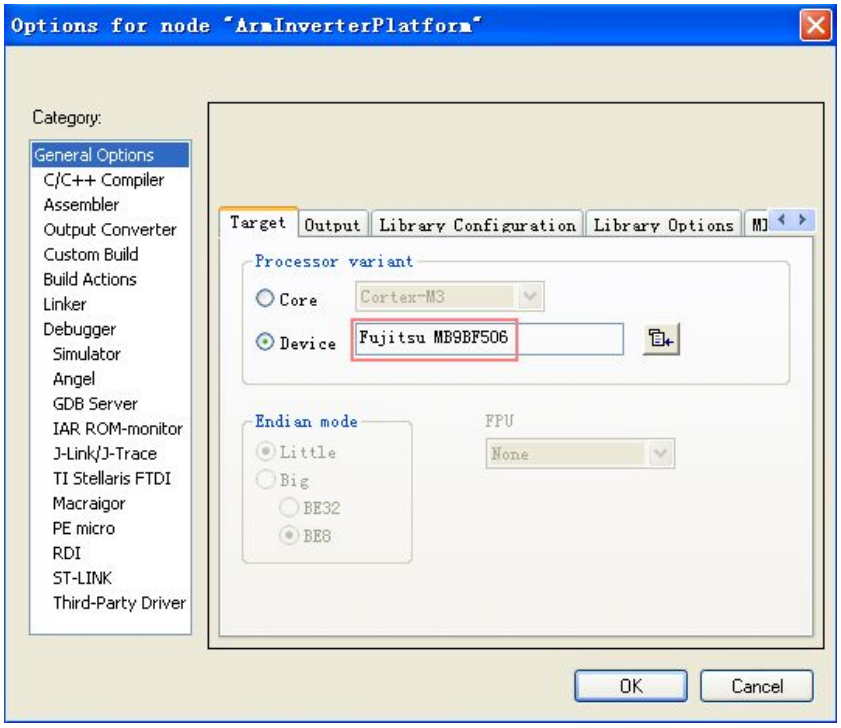


Figure 12 Target device selection: MB9BF506x

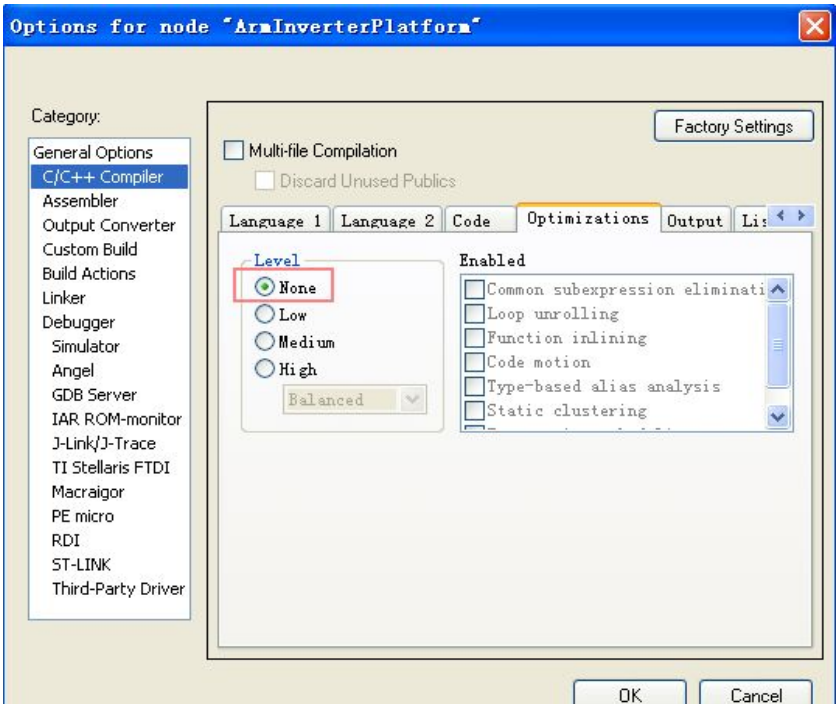


Figure 13 Optimization level: None

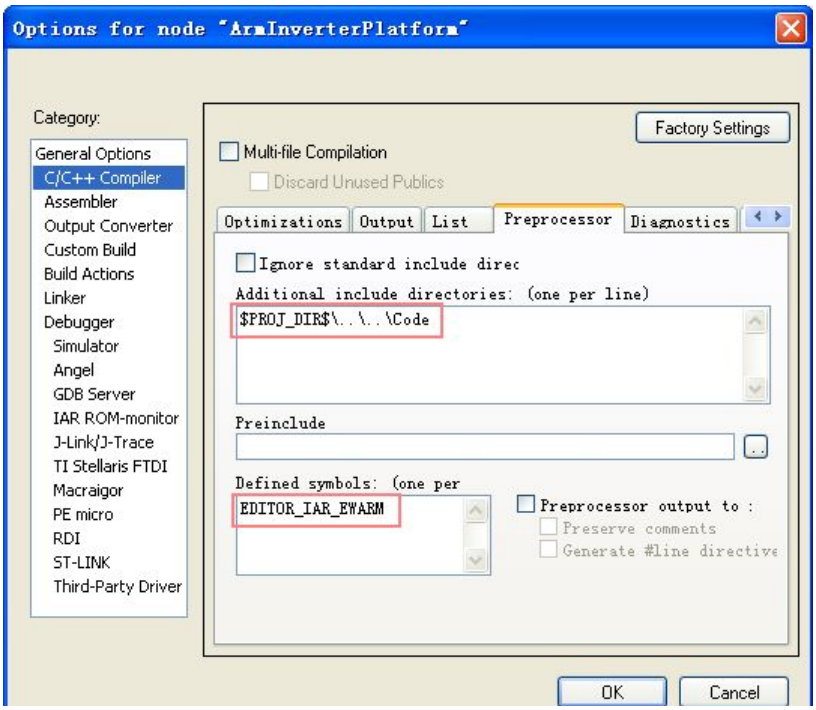


Figure 14 Preprocessor :above path and symbol should be set

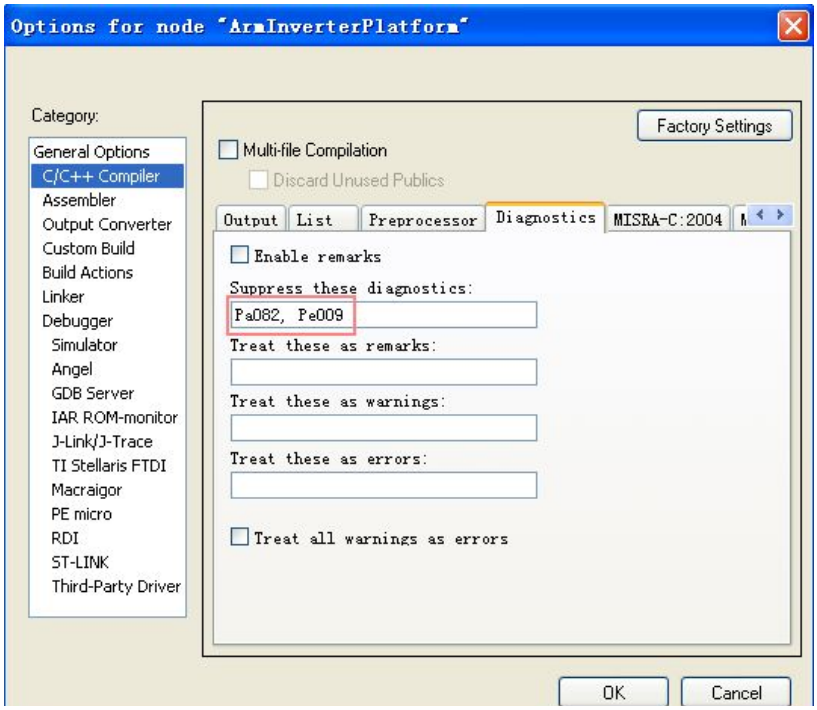


Figure 15 Diagnostics: Pa050, Pa082, Pa009,

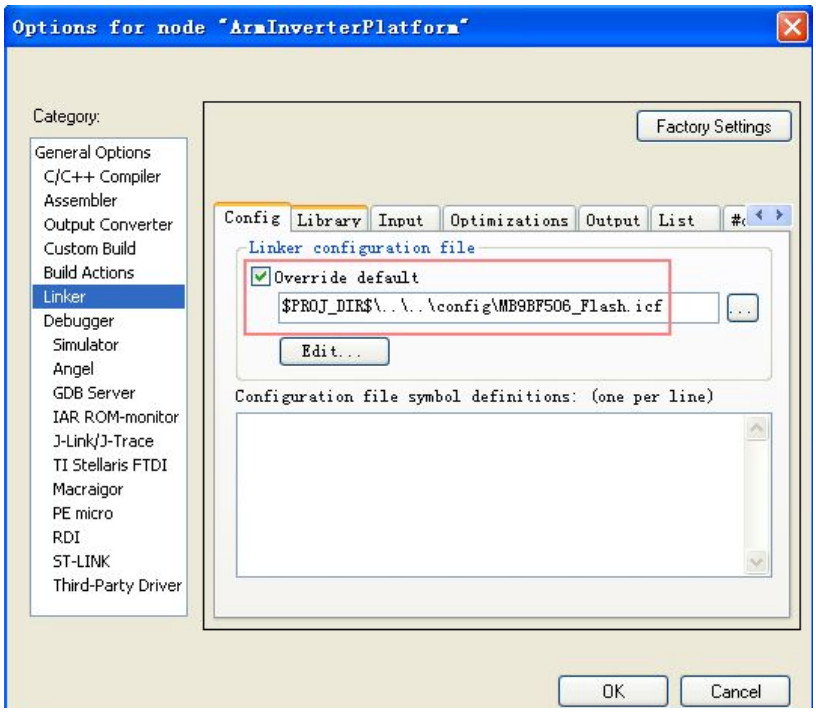


Figure 16 Linker configuration file: MB9BF506_Flash.icf

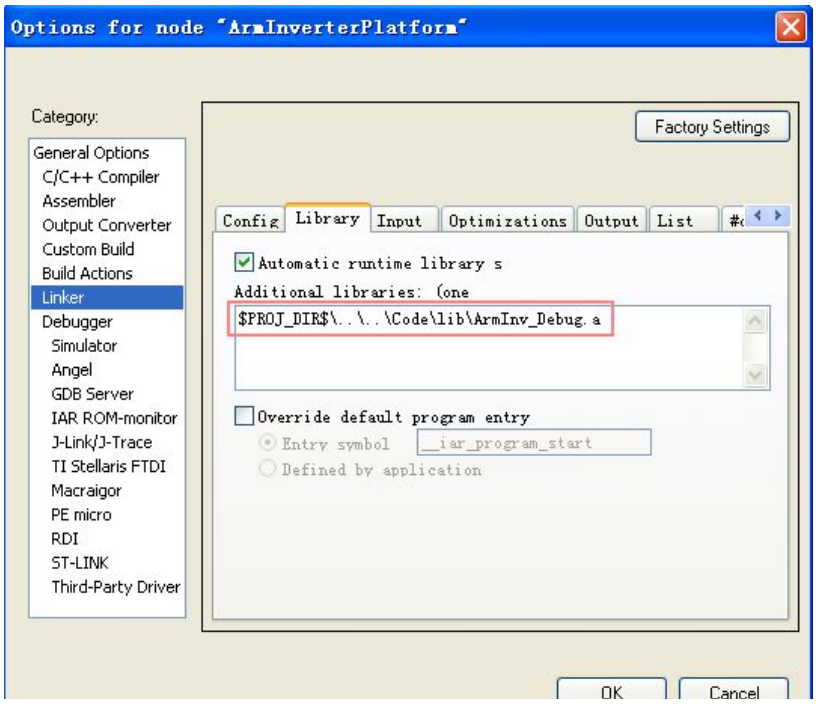


Figure 17 Library: ArmInv_Debug.a

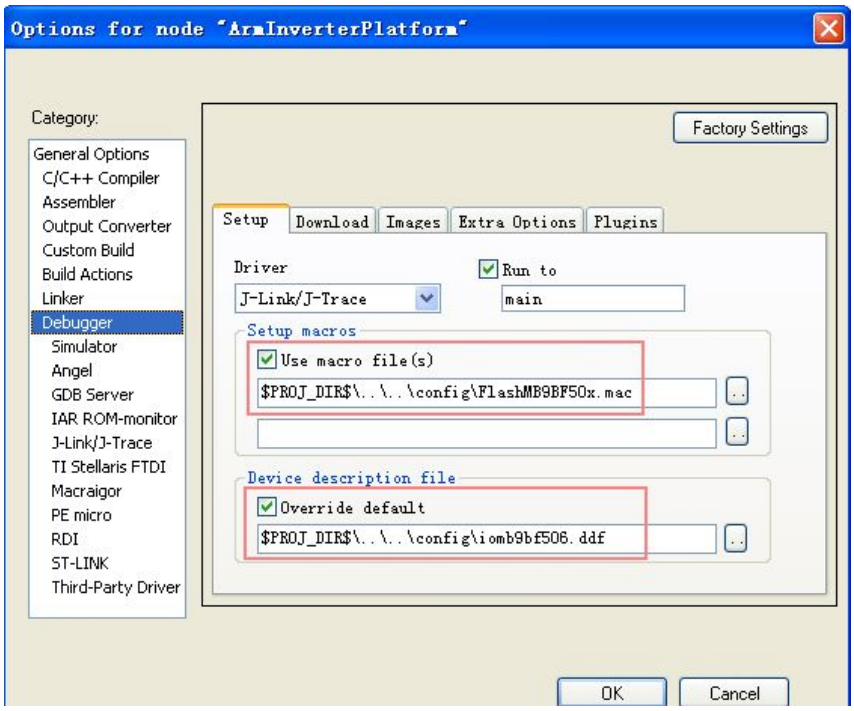


Figure 18 macro file and device description file: Flashmb9BF50x.mac and iomb9bf506.ddf

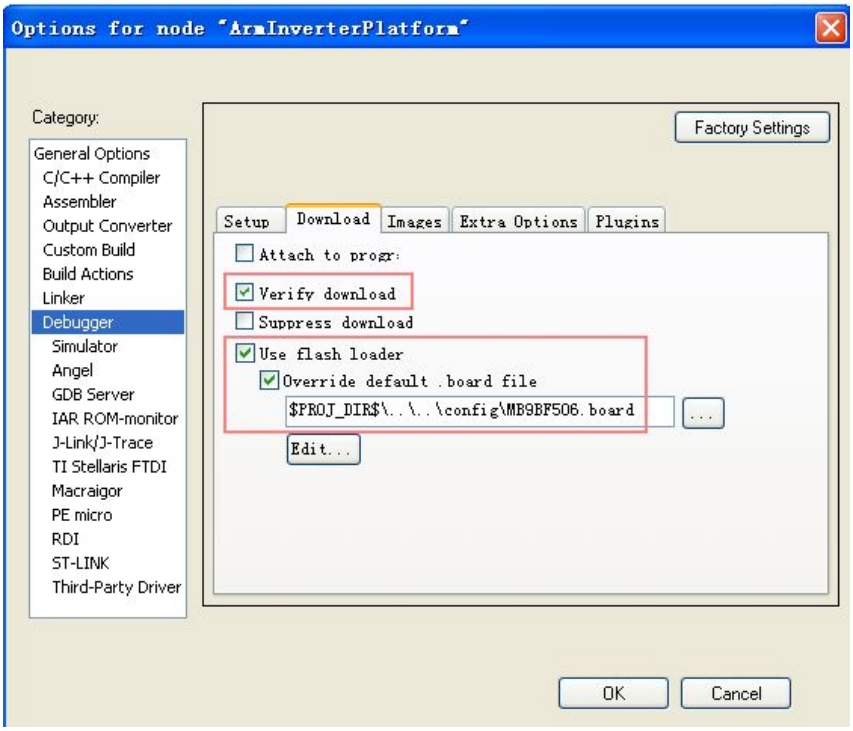


Figure 19 Flash download description file: some switch and Mb9BF506.board

9.2 Usage

This section introduces how to use MCL, include:

1. The structure of function main() and how to modify this function.
2. How to configure peripherals to meet user hardware.
3. How to add user routine into ISR of MCL
4. How to debug in EWARM
5. How to debug via GUI

9.2.1 main()

The coding of function main() is as follows. Also, below figure introduces the code,

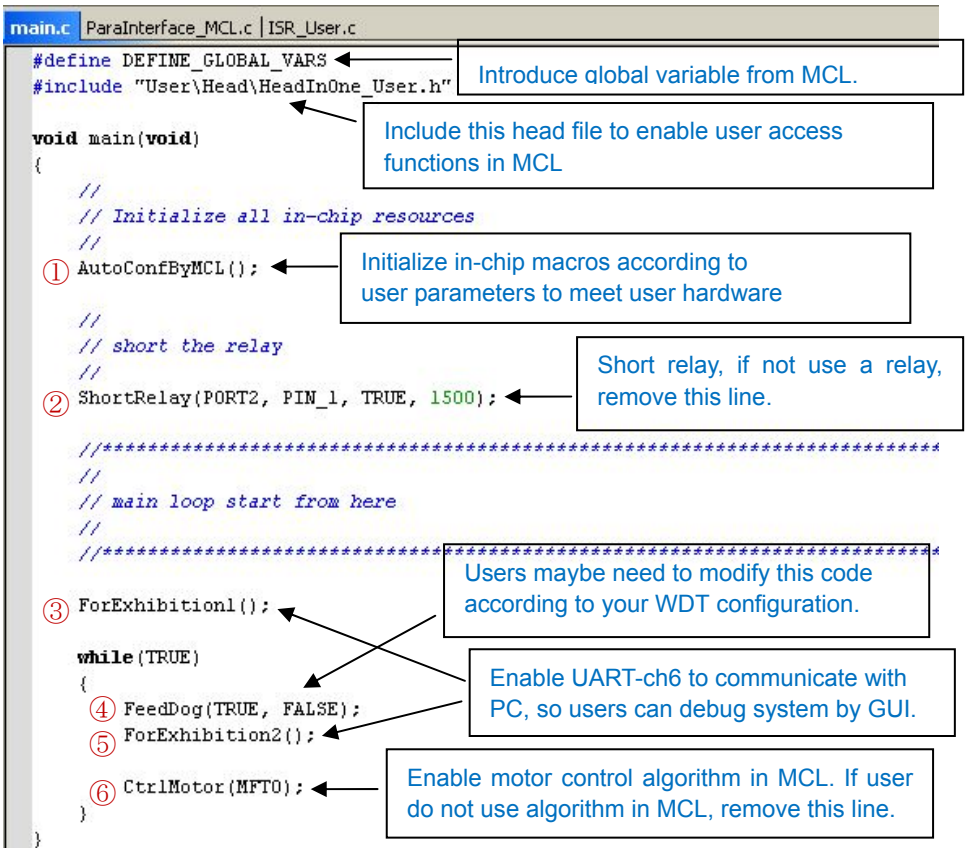

main.c
ParaInterface_MCL.c
ISR_User.c
#define DEFINE_GLOBAL_VARS
#include "User\Head\HeadInOne_User.h"
void main(void)
{
// Initialize all in-chip resources
① AutoConfByMCL();
// short the relay
② ShortRelay(PORT2, PIN_1, TRUE, 1500);
// main loop start from here
③ ForExhibition1();
while(TRUE)
④ FeedDog(TRUE, FALSE);
⑤ ForExhibition2();
⑥ CtrlMotor(MFT0);
Introduce global variable from MCL.
Include this head file to enable user access functions in MCL
Initialize in-chip macros according to user parameters to meet user hardware
Short relay, if not use a relay, remove this line.
Users maybe need to modify this code according to your WDT configuration.
Enable UART-ch6 to communicate with PC, so users can debug system by GUI.
Enable motor control algorithm in MCL. If user do not use algorithm in MCL, remove this line.

Figure 20 Introduction for main()

The code ① is the entry of on-chip macro initialization in MCL. If this code is deleted, code ⑥ cannot run correctly. So code ① and code ⑥ cannot be deleted to the users who want to use motor control algorithm in MCL.

If the users don't want to use motor control algorithm in MCL, code ① and code ⑥ can be removed. But it is recommend that only remove code ⑥, because code ① still can be used to initialize on-chip macros according to user configurations.

Code ③ and code ⑤ can be removed if users do not debug by GUI. But it is recommended to reserve it, because GUI can show some quantities as waveform. Some of them cannot measure directly by oscilloscope.

Code ④ maybe need to be modified. For example, if users configure to use soft watch dog timer instead of hardware watch dog timer, this code should be modified as FeedDog (FALSE, TRUE).

9.2.2 Configuring Peripherals

Please refer 7.2.

9.2.3 ISR Interface in MCL

In MCL, some interrupts are used, for example, FRT, DualTimer-ch2, UART-ch6, etc.

The ISR entry in MCL can be found in startup.c as follows:

```
// IRQ
ISR_DefaultHandler,                 // Anomalous Frequency Detection by C
ISR_SoftWatchdog,                   // Software Watchdog Timer
ISR_DefaultHandler,                 // Low Voltage Detector (LVD)
ISR_MFT_WFG,                        // Wave Form Generator unit0, unit1
ISR_DefaultHandler,                 // External Interrupt Request ch.0 to
ISR_DefaultHandler,                 // External Interrupt Request ch.8 to
ISR_DualTimer,                      // Dual Timer / Quadrature Decoder Un
ISR_DefaultHandler,                 // Reception Interrupt Request of MFS
```

Figure 21 ISR in startup.c

If users want to use motor control algorithm in MCL, please do not modify the entry name. Users that do not use motor control algorithm in MCL should modify this entry to meet your require.

User interface functions are embedded in MCL ISR to meet some users who want to do something in MCL ISR. The interface functions are in ISR_User.c. The user interface functions name are:

```
void DualTimer_User(void) ;
void MFT_FRT_User(void);
void ADC_Sampling_User(INT8U WhichADC) ;
void MFT_WFG_User(void) ;
```

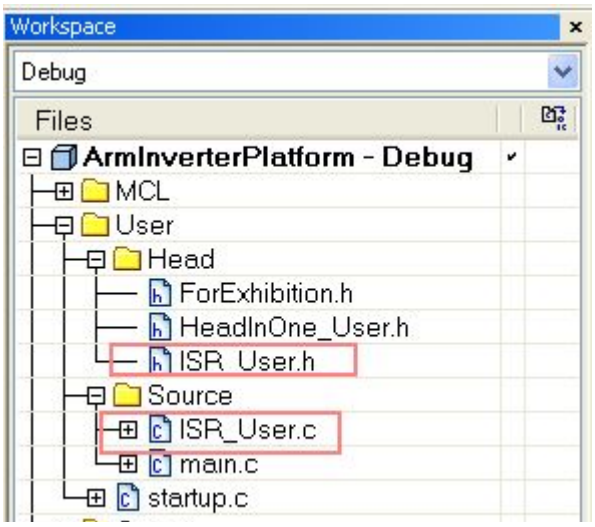


Figure 22 files of holding user interface in MCL ISR

main.c | ParaInterface_MCL.c | ISR_User.c

```
#include "User\Head\HeadInOne_User.h"

//
// Dual timer interrupt service for user
//
void DualTimer_User(void)
{ }

//
// FRT interrupt service for user
//
void MFT_FRT_User(void)
{ }

//
// FUNCTION:
// ADC interrupt service for user
//
// PARAMETERS:
// WhichADC - Which ADC unit interrupt.
//            Valid value is ADC_UNIT0, ADC_UNIT1, ADC_UNIT2.
//
// RETURN:
// none
//
void ADC_Sampling_User(INT8U WhichADC)
{ }

//
// WFG interrupt service in MCL
//
void MFT_WFG_User(void)
{ }
```

Figure 23 interface functions in MCL ISR

For example, if users want to do something in dual timer-ch2 interruption, user code can be added into the function void DualTimer_User(void). User code will be called after MCL ISR is executed.

9.2.4 Debug with EWARM

This section introduces how to debug motor control algorithm in MCL.

When coding is finished, select “download and debug” command to start debugging.

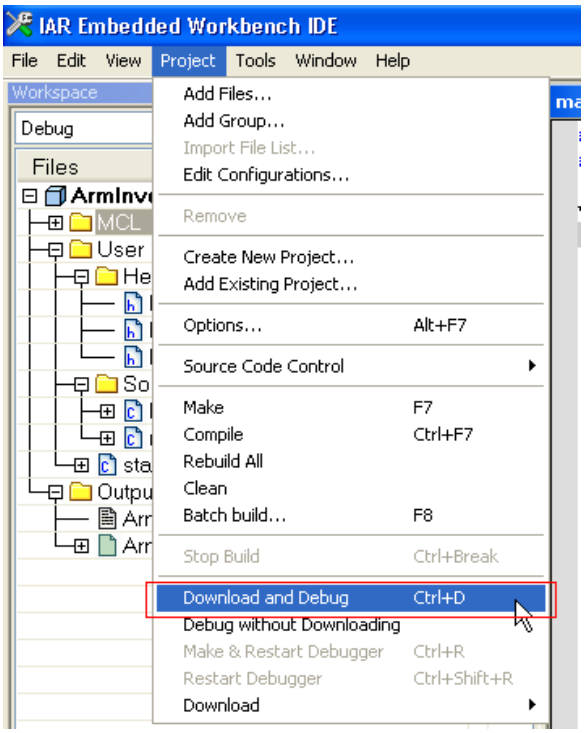


Figure 24 start debug window

In the debug window, please open “live watch window” and add global variable gstMotorRunCtrl into “live watch window”.

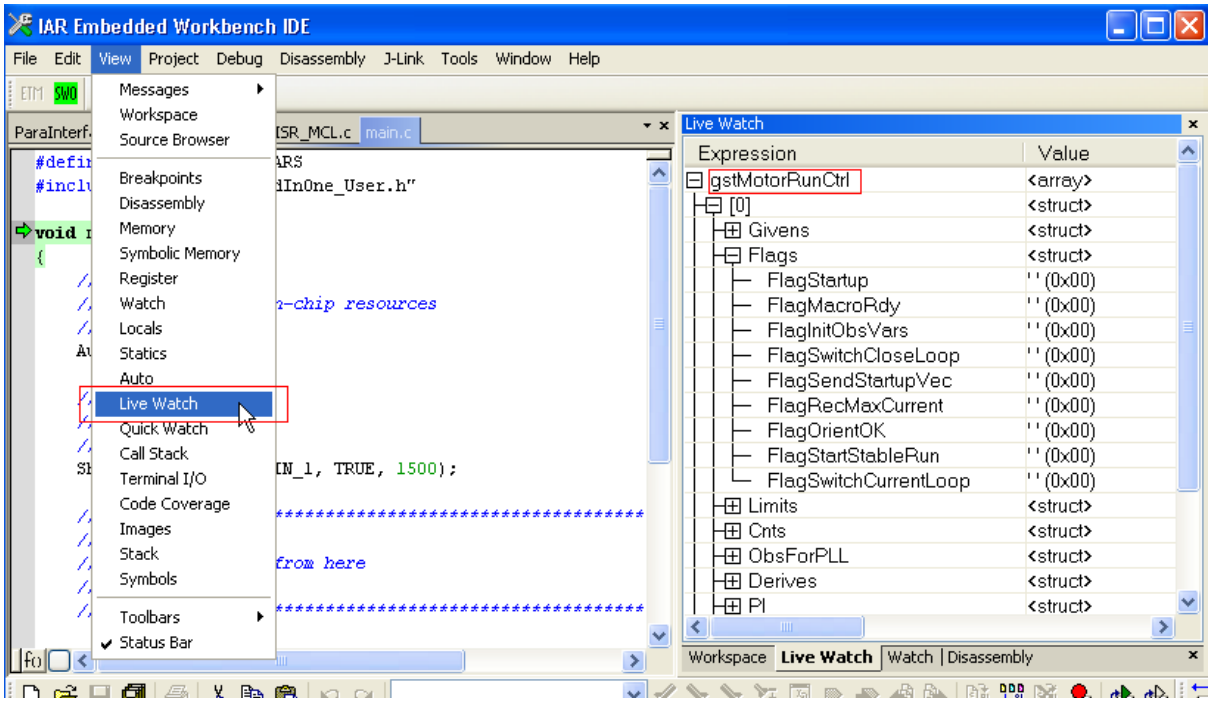


Figure 25 open live watch window and add global variable in

Expand “gstMotorRunCtrl → Flags”, some flags can be seen as follows.

Name	Description
FlagStartup	Motor startup flag. Modify this flag to startup or stop motor. 1 -- startup motor 0 -- stop motor.
FlagMacroRdy	To denote whether macro in-chip is configured successfully or not. 1 – macros in-chip is already configured 0 – macors in-chip is not ready.
FlagOrientOK	To denote whether rotor orienting is Ok or not. 1 – rotor orienting OK 0 – rotor orienting is not OK
FlagSwitchCurrentLoop	To denote whether motor is controlled in close loop. 1 – motor is controlled in close loop 0 -- motor is controlled in open loop

Expand “gstMotorRunCtrl → Givens”, some given values can be seen as follows.

Name	Description
ExpectW_CL_Q15	Motor running target speed (mechanical speed, unit is Hz). This value is Q15 format.
ExpectW	Motor running target speed (mechanical speed, unit is Hz). This variable is used for facilitate user input target speed.

Expand “gstMotorRunCtrl → Limits”, the motor speed limitation can be seen as follows.

Name	Description
MinRotSpeed_CL_Q15	Lower limit of motor running speed in close loop (mechanical speed, unit is Hz) . This value is Q15 format.
MaxRotSpeed_CL_Q15	Upper limit of motor running speed in close loop (mechanical speed, unit is Hz) . This value is Q15 format.

Expand “gstMotorRunCtrl → Derives”, some derived quantities can be seen as follows.

Name	Description
StepLenOfFreqChange_Q15	Motor speed change step, Q15 format.
CurrentSpeed_Q15	Motor current speed, Q15 format.
NextSpeed_Q15	Motor next speed, Q15 format.
AngleIncrement_Q15	Rotor angle increment, Q15 format.
→ RealValueFromSmpl → a_Q15	Real current value of phase-U
→ RealValueFromSmpl → b_Q15	Real current value of phase-V

Expand “gstMotorRunCtrl → PI”, PI regulator quantities can be seen as follows.

Name	Description
PI_W2Iqref	PI regulator for speed loop
PI_Idref2Vd	PI regulator for d axis current
PI_Iqref2Vq	PI regulator for q axis current

Expand “gstMotorRunCtrl → Output”, some output can be seen as follows.

Name	Description
CurrentRefInRotAxis	The current reference value in rotational coordinate
VoltageInRotAxis	The voltage value in rotational coordinate
VoltageInFixAxis	The voltage value in fixed coordinate

Expand “gstMotorRunCtrl → ObsForPLL”, some result come from motor control algorithm can be seen as follows.

Name	Description
InfoOfSmplCurr	Information relative to current sampling
CurrentInRotAxis	Current information in rotational coordinate
VoltageInFixAxis	Voltage information in fixed coordinate
BEMFInFixAxis	BEMF information in fixed coordinate
Wm_Q15	Mechanical speed (unit is Hz), Q15 format.
Theta_Q15	Rotor position (unit is degree), Q15 format.

9.2.5 Debug with GUI

GUI can get data from MCU and it can show data as waveforms r values on PC.
The advantage of using GUI, the users can see some of data which cannot measure directly by oscilloscope, For example, motor speed, BEMF, etc. These data may do help for users to debug system. We have supported many quantities in MCL, please refer to the software user manual.

- End -